Automatic Inference of Graph Models for Complex Networks with Genetic Programming

by

Alexander Bailey

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

COMPUTER SCIENCE

The Faculty of Mathematics and Sciences

Department of Computer Science

BROCK UNIVERSITY

June 21, 2013

Abstract

Complex networks can arise naturally and spontaneously from all things that act as a part of a larger system. From the patterns of socialization between people to the way biological systems organize themselves, complex networks are ubiquitous, but are currently poorly understood. A number of algorithms, designed by humans, have been proposed to describe the organizational behaviour of real-world networks. Consequently, breakthroughs in genetics, medicine, epidemiology, neuroscience, telecommunications and the social sciences have recently resulted. The algorithms, called graph models, represent significant human effort. Deriving accurate graph models is non-trivial, time-intensive, challenging and may only yield useful results for very specific phenomena. An automated approach can greatly reduce the human effort required and if effective, provide a valuable tool for understanding the large decentralized systems of interrelated things around us. To the best of the author's knowledge this thesis proposes the first method for the automatic inference of graph models for complex networks with varied properties, with and without community structure. Furthermore, to the best of the author's knowledge it is the first application of genetic programming for the automatic inference of graph models. The system and methodology was tested against benchmark data, and was shown to be capable of reproducing close approximations to well-known algorithms designed by humans. Furthermore, when used to infer a model for real biological data the resulting model was more representative than models currently used in the literature.

Contents

List of Tables

List of Figures

Preface

This thesis draws upon research of all kinds from many disciplines, although it most notably pulls from significant contributions made by M. E. J. Newman, A. Barabási, and D. Watts. Most readers should find it accessible, as there is no prerequisite knowledge of graph theory, complex networks, or genetic programming, although some knowledge of mathematics is assumed.

The following papers have been produced during the course of my MSc., and have arisen as part of this thesis work:

A. Bailey, M. Ventresca, and B. Ombuki-Berman, "Automatic generation of graph models for complex networks by genetic programming," in *Proc. of the 14th International Conference on Genetic and Evolutionary Computation Conference*, ser. GECCO '12 Philadelphia. New York, NY, USA: ACM, 2012, pp. 711–718.

A. Bailey, B. Ombuki-Berman, and M. Ventresca, "Automatic Inference of Hierarchical Graph Models using Genetic Programming with an Application to Cortical Networks," in *Proc. of the 15th International Conference on Genetic and Evolutionary Computation Conference*, ser. GECCO '13 Amsterdam. New York, NY, USA: ACM, 2013, to appear.

A. Bailey, M. Ventresca, and B. Ombuki-Berman, "Genetic Programming for the Automatic Inference of Graph Models for Complex Networks," in *IEEE Transactions on Evolutionary Computation.* revise and resubmit, January 23rd 2013.

Acknowledgements

To all those who belayed my journey and steadied me when I needed it the most, this is for you. Especially for my wife, Virginia, for her endless patience and encouragement. I would also like to thank my supervisors Dr. Beatrice Ombuki-Berman, and Dr. Mario Ventresca for their invaluable guidance, support, and inspiration.

Chapter 1

Introduction

The main contribution of this thesis is the introduction of the first automated approach to inferring graph models for complex networks. A *complex network* is defined as a collection of interrelated elements where the pattern of relations or connections between elements is meaningful; they are becoming an integral tool for our understanding of an enormous variety of natural and artificial systems [74]. For instance, a social network is a kind of complex network and the elements in the network are people and the connections between them imply friendship [86, 95]. We build networks like the internet to transmit signals [5, 99], or networks like a citation network to organize information [22, 42, 49, 76, 79], there are many networks found in nature like food webs, metabolic pathways, gene regulatory networks, and networks of neurons in the brain [4, 11, 26, 41, 43, 62]. Although real networks are most often very large and are not entirely regular, nor are they entirely random, making them difficult to analyze, it is the inherent meaning behind the pattern of connections rather than the size of the networks, or the difficulty of analysis, that makes them *complex.* The nature of complex networks is such that understanding their structure and dynamics *as a whole* is vital to understanding of the behaviour of the elements within them, as well as to understanding the behaviour of processing acting upon them. Unfortunately, our capacity to analyze complex networks is limited due to the prohibitive time complexity of many graph algorithms, the enormous size of many real networks, the frequent inability to collect data for whole networks, and the otherwise absence of many tools capable of providing meaningful information about such networks. For example, nearly three-hundred years of graph theory [24] have been aided by visualizations [68], and now with many networks consisting of millions or even billions of vertices (the internet, for example), searching for meaning in a visualization is an exercise in futility. Our frequent inability to properly study complex networks means that many systems are still not well-understood, despite their prevalence and importance in our lives. We require new tools for the study of complex networks that will enable us to better understand their structure, behaviour, and their functions.

To understand complex networks we must understand *why* the elements within the network interact, and to understand why they interact it is crucial to understand *how* they came to interact. If the processes that define how elements came to interact can be described as an algorithm, that algorithm, called a *graph model*, has an enormous number of applications. Graph models can be used to validate data sets, produce synthetic data on which to run simulations or test algorithms, infer how a network came to its current state, or check for the significance of connectivity patterns in real data. For example, in epidemics, human contact networks are constructed that provide information about how people in an environment interact with each other.

The human contact networks can be used to estimate how a disease will spread and at what rate, as well as help to devise effective vaccination schemes. However, reconstructing entire large networks from real data is often expensive, labour intensive, and time consuming [84]. Furthermore, it is difficult to interpret the data once it is collected. If a graph model could be derived that was capable of describing how human contact networks are structured, then it could be used to simulate unseen populations for which data cannot be collected, to extrapolate on existing population data, or to describe a contact network with only a few parameters. Furthermore, the algorithm itself can be analysed in such a way that many properties of the networks it generates can be computed in their limits.

Many human-designed graph models have been proposed over the last fifty years that reasonably reproduce specific real-life phenomena (see, for example [7, 21, 41, 43, 50, 79, 97–99]), and these models have been used to great effect in all manner of sciences, from diagnosing neurological disorders [9, 39, 67] to better understanding natural languages [27], revealing the structure of the internet [99], and even to better understand the essential processes of life itself [11, 44, 93]. However, the relationship that graph models have with the complex networks they model is necessarily a cyclical one: we need to understand a complex network in order to model it in order to understand it. Therefore, the act of crafting an accurate graph model is a very time intensive and challenging task that depends heavily upon the ingenuity and knowledge of the crafter. Naturally, reusing existing models has become preferable to deriving a model for new kinds of data, sometimes at the risk of using an inaccurate model, and a large number of researchers have dedicated their time to the task of classifying different complex networks such that they may be modelled, at least somewhat effectively, with existing graph models (see, for example [1,2,4,5,8,13,26,27,62,67,78,88]). However, the decision as to which graph model should be used is sometimes non-trivial, e.g., with some brain networks [101], and once a model is selected parameters may need to be tuned in order to use it effectively. Thus, we have arrived at the major motivating factors for the work contained in this thesis:

1. Complex networks are ubiquitous, and understanding them will necessarily lead to a greater understanding of our universe.

2. Graph models are an indispensably useful tool for the analysis and understanding of complex networks.

3. Graph models are non-trivial to design, because complex networks are hard to understand.

4. It is not obvious how to select an existing graph model to simulate a given network.

5. It is not obvious how to tune a graph model's parameters when it is selected.

6. Most importantly, the process of creating graph models has always been done manually.

1.1 Main Goals

This thesis proposes the first ever methodology for the automatic inference of graph models for complex networks, and contends that by automating the process of graph model construction the challenges of designing, selecting, and tuning a graph model can be overcome. An automated approach to the construction of graph models would significantly reduce the amount of human effort required to create a model, and would be particularly useful in cases where existing graph models do not exactly fit the features of the network or networks at hand. The ease of designing a graph model robust to more specific or exotic networks would eliminate the need to select or tune an existing model where human efforts have failed to produce accurate models. Furthermore, automatically generated models may out-perform human attempts at constructing graph models, especially when a good model may exist but remains dubious to human designers because the network at hand has a large number vertices or edges making it difficult for a human to identify patterns in vertex connectivity. The utility of an automated approach is undeniable, providing it is capable of generating accurate and meaningful models.

Secondly, this thesis proposes the first ever application of genetic programming (GP) for the task of graph model inference. Genetic programming, inspired by population genetics and Darwin's theory of evolution, "evolves" populations of programs representing solutions to a problem, by allowing the most fit to mate while the least fit die out – the best solution in the population after some number of generations is saved. Genetic programming is a natural fit to this problem because the goal here is to reproduce an algorithm that grows a structure, an application at which GP excels [54, 77]. Genetic programming has also been shown to produce good results in areas where the domain is not well understood [52, 53]. Further motivation for the use of GP results from the multi-objective nature of the problem, whereby multiple graph structure properties must be simultaneously considered during the design process. There is a large body of work concerning multi-objective strategies for Evolutionary Algorithms (EA) [15, 102].

1.2 Challenges and Contributions

The application of GP to the task of automatically creating graph models is non-trivial, especially given that there is no prior work on this problem. First, it should be noted that there are many different types of networks: dynamic, static, weighted, unweighted, directed, undirected, etc. Each network type has its own unique challenges and to design a methodology capable of deriving models of each type is far beyond what could be accomplished within the scope of this thesis. Therefore, the focus of this thesis is on undirected, unweighted networks, which are very general and find applications in many areas. Three major challenges exist in the design of a GP methodology for inferring graph models of any kind, however the comments here are made with undirected, unweighted, networks in mind. The first: how should a fitness function be defined? The second: how can a GP language / function set be effectively designed? The third: how do we know the results are meaningful?

1.2.1 Fitness Function

Genetic programming relies on an ability to gauge the fitness of a program for its purpose. In the case of evolving graph models, the GP needs to know how well a program in the population approximates the underlying process that created the network at hand. If this information was known *a priori* then GP would not be needed to construct a graph model. However, the information required should be somehow encoded into the network of interest, which must be somehow compared to the programs in the GP population. Of course, a network cannot be compared to a program, so the programs in the population must be used to generate networks to which the network of interest must be compared. Now this has become a problem of graph comparison, however, the goal is not to reproduce a specific network (this is *not* an isomorphism problem), but rather to reproduce the underlying processes which created that network, and those processes are likely capable of generating a large number of different graphs. This thesis proposes that statistical measures of the large-scale structure of the network are sufficient to construct a fitness function, as evidenced by empirical studies of such measures and their ability to predict the behaviour and dynamics of processes enacting on complex networks [6, 94], as well as their continued use in classifying different types of complex networks.

1.2.2 Function Set

Genetic Programming requires a set of functions related to the task at hand in order to construct meaningful programs for the population. Since no prior work could be drawn from to derive a function set it was not obvious how to do so. The function set must be of sufficiently high-level to express meaningful relationships, allowing GP to not become bogged-down in its search for useful constructions, while being expressive enough that a large variety of models can be created. This thesis proposes the first GP language for the construction of graph models for complex networks.

1.2.3 Validation

Once the GP system is designed and a graph model generated, it cannot be assumed that the model is meaningful or useful. To validate the system and gauge its ability to infer meaningful models, it was first used to re-create human designed models from benchmark networks created by the Erdös-Rényi, Watts-Strogatz, and Barabási-Albert models, three well-known and very different graph models. Each benchmark model was designed by humans and each one represents a significant advancement in our ability to understand and study complex networks. The resulting evolved models were sampled and the networks generated by the evolved models were rigorously compared to the benchmark networks. More importantly, the evolved algorithms could be directly compared to the known algorithms that generated the benchmark networks – GP proved very capable of creating close approximations to the human-designed models.

To further evaluate the GP approach, it was used to infer a model for a real-world network common in computational neurology literature, which represents a cat's cortex. The cortical network is interesting because there has been some disagreement as to which graph model is best suited to approximating the data, and at least one author that found commonly applied graph models to be insufficient [101]. The cortical data also possesses a common organizational feature of networks in which some groups of vertices are more densely interconnected than others, and is said to exhibit *community structure*. With some small extensions to the original methodology, the GP was able to create models capable of accounting for the community structure of the cortical network, and outperform commonly used human designed models.

1.2.4 Contributions

In summary, the following major contributions are made by this thesis:

1. Proposes the first ever methodology for the automatic inference of graph models for complex networks.

2. Proposes the first ever application of GP to the problem of automatically inferring graph models for complex networks.

3. Proposes a fitness function suitable for evolving graph models for complex networks.

4. Proposes a GP function set for the construction of graph models for complex networks.

5. Proposes a methodology for automatically inferring graph models exhibiting defined community structure.

1.3 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 provides a brief summary of the study of networks over the years, and takes an intuitive tack at familiarizing the reader with some real-world networks and the concept of a graph model. Chapter 3 provides the tools necessary to understand and analyze complex networks as graphs within the context of this thesis. Having the requisite tools, Chapter 4 describes some common graph models that have been used to model real-world systems and have been used to generate benchmark data, or have been the source of inspiration when designing the GP function set. Chapter 5 gives a short overview of GP and details its workings, leading into Chapter 6 which describes the exact methods used by the proposed GP system. Chapter 7 shows how the system was used to reproduce some of the models discussed in Chapter 4 and Chapter 8 shows how the system was used to infer a model for the cortical structure of a cat. Finally, Chapter 9 offers some conclusions and reflections on this work as well as motivation for future work. Additionally, for interested readers, there are three appendices A, B, and C, which describe some of the early results of this work, as well as expanded results not shown in Chapter 7 and 8.

Chapter 2

Complex Networks and Graph Models

Complex networks are everywhere, they often arise naturally, and as such the processes which govern their behaviour and structure are usually decentralized. Their decentralized nature has two implications. First that their structure is not always obvious because it is emergent from a collection of interrelated things, and second, that an understanding of the rules governing how elements interact reveals an understanding of the structure as a whole. It may be expected that networks of the same kind can be characterized in the same way, e.g., two different groups of people might have similar patterns of socialization. Perhaps more interesting is that two networks of apparently different kinds can also be characterized in the same way, e.g., socialization patterns of children and the Internet. Observations of this kind have led to a push to identify properties common to all or many networks, and a few prevalent features have been the focus of a large number of studies. Various properties of complex networks will be defined and discussed in the next chapter. This chapter provides some context for the modern study of networks by giving an overview of the study of networks through the years. This chapter also provides a vehicle to familiarize the reader with a number of real-world networks and the categories to which they belong.

2.1 Complex Networks

The study of networks is intimately related to the field of graph theory, which has formed the foundation for observations and analytical treatment of various networks that have been the object of interest across different disciplines. However, many of the questions posed by traditional graph theory consider only small groups of vertices in practice, which is impractical relative to the millions of vertices in many real-world networks [68]. It has become necessary to find ways to characterize and study the structure of large networks, shifting the focus away from traditional graph theoretic approaches, although the new approaches are deeply rooted in graph theory. The new forms of complex network analysis are typically statistical in nature [68] or look for the significance of connections relative to a null model of random connections, and have been able to shed some light on the structure of very large networks.

The social sciences have had a long standing history of studying networks, and some of the earliest studies of real networks come from that discipline [30]. The social sciences are naturally concerned with the interactions of people, *social networks*, in which the elements are people and two elements are linked if they interact. Early studies of social networks appeared in the 1930s, a notable study by Moreno focused on the friendship patterns of small groups of children [63]. Moreno observed patterns of association related to the number of friends people already had, similar levels of motivation, etc. The study was important because it marked a new era of social study through networks, Moreno himself introduced the *sociogram*, a graphical representation of a network of people, and the macroscopic study of social phenomena called *sociometrics* revealed through his sociograms. Another famous study, focused on social patterns between a group of 18 women in a community in the Southern United States [18]. The study reports that the women had organized themselves into two distinct groups, and by looking at maps of their interactions, it was possible to determine which person played a core or peripheral role in those groups, based on the kinds of events they attended. The "southern women" network has also known to possess some interesting properties and details in its organization that are not obvious, despite the data set being small [32]. The southern women study continues to be important today because it is an excellent example of the complexity and subtleties of the organization of even small networks. The southern women study also posed a precursor to the problem of giving the terms "group" and "position" an exact definition in the context of a social environment which saw nearly everyone interact with each other [32]. Finally, an important study by A. Rapoport should be mentioned, as he may have been the first theorist to stress the importance of the degree distributions in all kinds of networks, not just social networks [68, 80].

These developments have come to play a crucial role in the modern study of networks and their organization. However, early studies often suffered from problems of inaccuracy or bias, this was perhaps unavoidable given that the primary method of social data collection was by interview or survey [74]. Once the challenges of data collection were realized, many studies shifted their focus to more reliable sources of data, such as co-authorship networks. Co-authorship networks can be reconstructed from publication records, are a kind of collaboration network (which is itself a kind of social network), and are usually expected to be accurate (because people take authorship seriously) [68]. Co-authorship networks naturally bring to mind a related kind of network, a network of publications, in which two publications are linked if one cites the other. This is a new kind of network, and brings us to our next category.

Information networks are typically networks of abstract things and the links between them represent the sharing or flow of information. Citation networks are a good example of information networks which possess many interesting properties (interesting enough that the study of citation networks has a name, *bibliometrics*) [22]. Citation networks were an excellent source of reliable data for early studies of networks, and have been the basis for some important observations. For instance, work by Price showed that papers in a citation network which already had some citations were more likely to be cited than those papers which had fewer citations. He called this property *cumulative advantage* [79], and although Price made some vague references to other "social phenomena", he himself focused only on the role of cumulative advantage in citation networks. Although, his ideas served as a catalyst for the study of other networks of information in which the same phenomena can be observed. For example, Price's phenomena is also observed in the network of co-occurrences of words in sentences [27], as well as the number of hyperlinks a webpage on the World

Wide Web (WWW) will have pointing to it [42]. The WWW is of course, related to another network of a different category, called the Internet.

Technological networks, are typically made up of machines that pass signals or commodities. Power grids, for example, have been studied in the past [97]. Airline routes [5], are another example of a technological network. Distribution and collection systems also fall into this category [74], and so some natural systems also fall into this category such as river systems. A network of particular significance is the Internet, the Internet differs from the WWW, as it is not a conceptual network of information but a physical network of computers and other devices that are linked together by physical (or maybe wireless) connections [99]. The Internet naturally has a hierarchical structure, Network Backbone Providers (NBPs) connect Internet Service Providers (ISPs) that finally connect end users. However, the specific structure of the Internet is maintained by different entities, and there is no central authority that knows routes between autonomous systems, this is handled by a reachability protocol called the Border Gateway Protocol (BGP) [74].

We have seen that technological networks, such as the internet, are capable of operating in a decentralized way, and the same is true of many biological systems. *Biological networks* are the final category of networks, and in this category exists many fascinating systems. For example, metabolic pathways, which are a series of chemical reactions within a cell that are catalysed by enzymes and are responsible for necessary cellular functions [44]. Another example of a biological network is a Protein-Protein Interaction (PPI) network, the network is formed by proteins, which interact with each other physically "folding" each other into shapes that define their function [11].

Many of the networks mentioned require specific properties to function properly. Communication networks require short routes between points in the network for efficient signal transmission, as well as a relatively small number of links between points because of the costs associated with creating links. It is not hard to imagine that biological networks may have similar constraints, i.e., there are many biological communication networks and it costs energy for biological systems to produce anything. Many networks require fault tolerant structures, robust to the failure of random or targeted attacks on network infrastructure, such as in communication networks or gene regulatory networks. These factors manifest themselves in similar ways across networks with entirely different functions, such as multiple alternative short routes between any two points [97], or the presence of highly connected "hubs" within the network [7]. Vertices with common functions or that require more frequent intercommunication will often arrange themselves into tightly knit "groups" which are less connected to the rest of the network than they are with each other [73]. Figures 2.1, 2.2, 2.3, and 2.4 show plots of a social network, an information network, a technological network, and a biological network respectively. The reader is invited to examine the figures for evidence of interesting structures and structures common to more than one network.

Perhaps the most fascinating thing about the organization of the networks mentioned in this chapter, is that they have no central planning agent. Network organization arises naturally from the function of each element within the network as the network grows. The processes governing the growth and dynamics of complex networks have therefore garnered an increasing amount of attention in the literature, as an understanding of these processes constitutes a much deeper understanding of the network as a whole [7, 37, 50, 56, 74, 93, 97]. As a side effect to learning the underlying processes governing the behaviour and structure of networks, algorithms called *graph models* can be created, which are capable of constructing simulated networks with behaviour and organization similar to real networks of importance.

2.2 Graph Models

A *graph model* is an algorithm designed to simulate the processes which govern the connection patterns between elements in real networks. Graph models are usually probabilistic in nature, and capable of producing an infinite set of networks, all similar to each other in some way (Figure 2.5 illustrates this process). If an algorithm can successfully be designed, which produces graphs that share similar properties with a real-world network then we can say we have a graph model for that network. Note that it is not necessary that the model reproduce the exact network it is modelling. In fact, it is preferable that it does not do this. A real-world network is most likely one instance of how such a network could form, graph models provide information about *all* networks with the same growth dynamics.

Graph models, can be used to simulate large networks when collection of real data is not feasible due to cost, or technical limitations, or predict how a network will change over time. If a collection of networks are simulated, the impact of destructive or diffusive processes on modelled networks can be explored. Graph models can also provide useful null models that can be used to check for the significance of patterns in real data. Furthermore, the models can be used to validate large noisy data sets, or to solve for behaviour in the limit of large networks.

The genesis of network modelling is usually associated with a 1950's publication of the *Random Graph Model* by Erdös and Rényi [23], and the random graph was used to inform many studies of network organization until the publication of the *the Barabási-Albert* (BA) model [7] and the *Watts-Strogatz small-world* (WS) model [97]. The BA model and the WS model characterized some seemingly universal properties of a large number of real networks which could not be explained by the random graph. Since then, a variety of models have been proposed [37, 41, 50, 56, 93] and used to study the behaviour of many networks with great success. For example, graph models have been used to study the spread of disease [6], assist in the diagnosis of degenerative brain diseases [20, 90], characterize brain activity in patients with epilepsy [67], simulate the effect of white-matter lesions on the brain [45], discern the organization of cortical structure in mammalian brains [83], better understand the function of interacting proteins [93], explore the topology of the internet [99], scientific collaboration networks [8], metabolic networks [26], food webs [62], and many more [74].

For more than fifty years humans have been constructing graph models manually, and they have proven to be useful tools. However, a significant amount of human

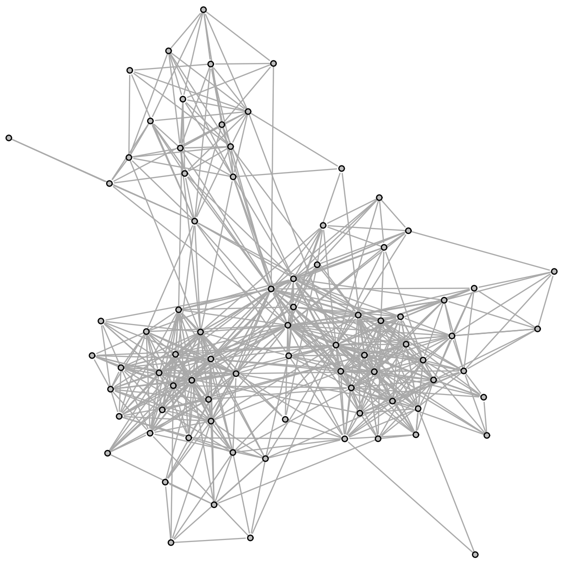


Figure 2.1: The personal friendship network of a faculty of a UK university [65].

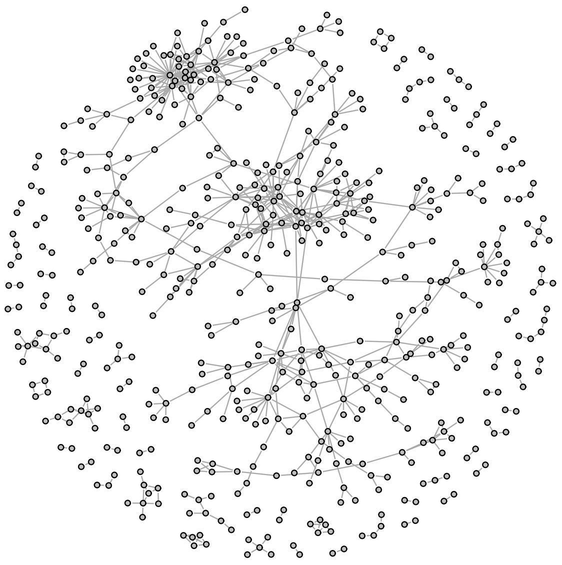


Figure 2.2: Citations between papers in high energy physics between Jan. 1992 and June 1993, isolated from data published in [55].

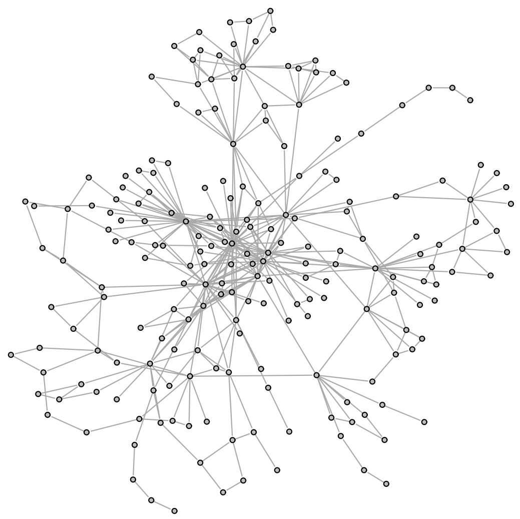


Figure 2.3: Airline routes between Canadian airports, network reconstructed from data in the OpenFlight database [92].

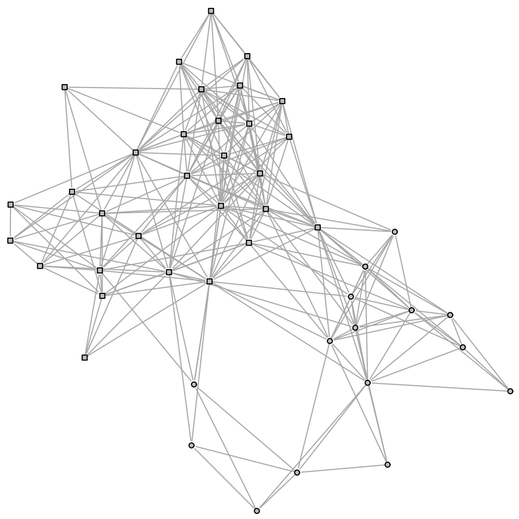


Figure 2.4: Connections between cortical areas of a Macaque monkey (one vertex represents a large number of neurons) [75].

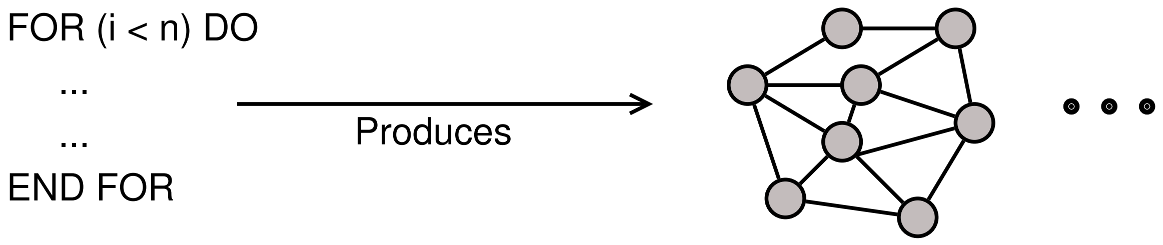


Figure 2.5: A *graph model* produces a set of graphs

effort and domain knowledge is required to construct graph models accurately. Furthermore, it is also not always clear how to select a model for a given network. Automatic construction or inference of graph models contributes a solution to both challenges. Providing researchers with a method for quickly creating accurate graph models for any real-world network could facilitate a deeper understanding of many complex systems, leading to more of the important discoveries existing graph models have afforded. The remainder of this thesis will focus on how to facilitate the automatic inference of graph models, the challenges encountered while seeking a solution, and some necessary and thorough results that serve as validation for the proposed mechanism. Following the initial analysis some exciting results are presented, showing the performance of the system on real-world data. However, some basic definitions of graph theory and properties of networks must first be given in the proceeding chapter.

Chapter 3

Measurable Properties of Networks

This chapter provides an overview of many useful measures that can be used to quantify network structure. The measures permit inferences concerning the structural or functional behaviour of the network, while also providing a means to ascertain the dis/similarity of two networks by the presence or absence of particular properties. While some of the properties come directly from graph theory, some have also found their way into the study of networks from physics and the social sciences. The study of real-world networks is most usually complimented by measures which consider the large-scale dynamics of the entire network, and come in the form of statistical measures.

There are a very large number of results and properties related to networks. Only those relevant to this thesis will be discussed here, beginning with defining graph representations in Section 3.1, followed by some basic graph properties in Section 3.2, and finally descriptions of the large-scale organization of networks in Section 3.3. A comprehensive overview of network properties can be found in [74].

3.1 Representation

A graph is the mathematical abstraction of a network, and a graph G is defined such that $G = (V, E)$ where V is the vertex set and E is the edge set, the size of the vertex set is $|V| = n$, and $|E| = m$. Graphs can be represented in a number of different ways, and each way has some advantages and disadvantages. For mathematical treatment they are typically represented as an *adjacency matrix*, which is an $n \times n$ matrix of values in which the value at the i^{th} row and j^{th} column is non-zero if vertex j is connected to vertex i (or more succinctly *iff* $(i, j) \in E$) [74]. Graphs can also represented as an *adjacency list*, which is a list of lists where the i^{th} list will contain j if vertex i is connected to vertex j.

Graphs may also be represented as an *edge list* which is a list of tuples, where a tuple (i, j) is in the list if vertex i is connected to vertex j. The latter two are sometimes used for digital representation because they require less space (linear in terms of edges and vertices versus quadratic with respect to vertices). However, adjacency matrices are more intuitive and more convenient to manipulate using matrix operations on paper. Fig. 3.1 gives an example of an adjacency matrix, and the corresponding graph is shown Fig. 3.1.

Some observations can be made about the adjacency matrix in Fig. 3.1, it is symmetric, the values are binary, and the diagonal consists of all zeros. This kind of adjacency matrix corresponds to *simple graphs* [74]. If the diagonal contained non-zero elements they would correspond to a *self-edge* or *self-loop* in which vertex

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}. \qquad A_{ij} = \begin{cases} 1 & \textit{if there is an edge from } j \textit{ to } i \\ 0 & \textit{otherwise} \end{cases}$$

Figure 3.1: An adjacency matrix.

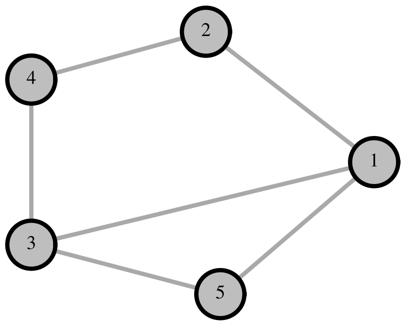


Figure 3.2: The graph corresponding to the adjacency matrix in Fig. 3.1

i connects to itself [74]. If the elements were not binary the values in the matrix could correspond to multiple edges connecting i and j, called *multi-edges*, or possibly a graph which had weights associated with the edges, a *weighted graph* [74]. If the adjacency matrix was not symmetric then it would correspond to a graph in which the edges were not bi-directional, called a *directed graph*, meaning i may be connected to j without requiring that j be connected to i [74]. If a graph contains self-loops, multi-edges, weights, or directed edges it is not simple [74].

Undirected, unweighted, graphs are sufficient to capture the structure of a large number of networks, and often a relaxation from a directed graph to an undirected one can reasonably be made in cases where the majority of associations between vertices are bi-directional. For purposes of this thesis only undirected, unweighted, graphs are considered.

3.2 Properties and Measures

This section will provide some basic definitions and properties of graphs which can be used to describe facets of a network, and upon which a greater understanding of networks can be built. As with the rest of this thesis, properties are given with respect to undirected unweighted graphs, which is the focus of this thesis.

3.2.1 Degree

Perhaps the most basic, albeit, very important property of a vertex is its *degree*. The degree of a vertex in an, undirected and unweighted graph corresponds to the number of edges connected to it. Given n vertices in such a graph, the degree k_i of vertex i can be written in terms of the adjacency matrix A as [74]

$$k_i = \sum_{j=1}^{n} A_{ij}. \tag{3.1}$$

The number of edges m in the graph can also be expressed in terms of the vertex degrees. Given that there are two ends of an edge for every node the number of edges is

$$m = \frac{1}{2}\sum_{j=1}^{n} k_i, \tag{3.2}$$

and the *average vertex degree* c is computed as

$$c = \frac{1}{n}\sum_{j=1}^{n} k_i = \frac{2m}{n}. \tag{3.3}$$

It is also useful to note that the maximum possible number of edges in a simple graph is $\begin{pmatrix} n \\ 2 \end{pmatrix} = \frac{1}{2}n(n-1)$, i.e., the number of all two-tuples $(i, j) \in E$ where $i, j \in V$ and $|V| = n$.

3.2.2 Paths

A *path* in a network is made up of a sequence of vertices $P = (p_1, p_2, ..., p_m)$ such that for every consecutive pair of vertices p_{i-1} and p_i there is a tuple $(p_{i-1}, p_i) \in E$. That is to say, there is an edge joining consecutive vertices in a path. The *length* of a path P is the number of edges traversed or $|P| - 1$. Generally speaking, a path can revisit edges or vertices, or intersect itself, however, a path that does not do these things is called a *self-avoiding path*.

A specific kind of self-avoiding path, called a *geodesic path*, or path of shortest length between two vertices (a path can always be made shorter by removing a loop or not backtracking) is of particular interest [74]. Geodesic paths play important roles in communication networks or networks on which diffusive processes are at work. Geodesic paths are not always unique, and it is simple to construct a graph which has more than one geodesic path between two vertices. See Fig. 3.3.

It is often useful to characterize the network by the *average geodesic path length* or *characteristic path length* [97], which describes the average communication or distance of travel between any pair of vertices. The average geodesic path, l, can be computed

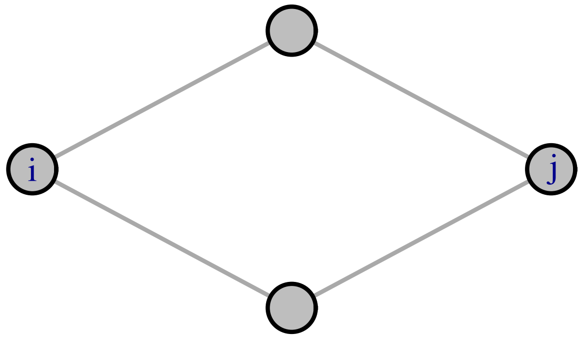


Figure 3.3: A graph with multiple geodesic paths from i to j.

as [74]

$$l = \frac{2}{n(n+1)} \sum_{i \geq j} d_{ij}, \tag{3.4}$$

where d_{ij} is the geodesic path length between vertices i and j in an undirected graph.

A specific geodesic path, called the *diameter* of the network, is defined as the longest geodesic between any vertex pair i and j. It is important because it provides an upper bound on the length of communication pathways between points in the network

Geodesic path lengths can be found by traversing the network using a breadth first search (or any search algorithm). Search algorithms will not be discussed here, but an excellent overview of search algorithms can be found in [82], or see [74] for a collection of algorithms relevant to graph searches.

3.2.3 Betweenness Centrality

The concept of *centrality* is used to describe how important a vertex may be within a network [74]. There is more than one way to decide how important a vertex is within a network, and a commonly used measure of centrality considers how many geodesic paths pass through a vertex i, and the count is normalized by the number of geodesic paths in the network. This measure of centrality is referred to as *betweenness centrality* [31]. Vertices with relatively high betweenness centrality values can have a significant influence on the flow of information within a network given that messages will usually be ushered along the shortest path between any two points. The removal of nodes with high betweenness therefore can be very disruptive to network communication. Consider the large vertex in Fig. 3.4. It necessarily has high betweenness because it controls the flow of information between two large clusters of vertices, removing it would disconnect those clusters from each other and block communication. Betweenness centrality x_i for a vertex i is computed as

$$x_i = \frac{1}{n^2} \sum_{st} \frac{n^i_{st}}{g_{st}}, \tag{3.5}$$

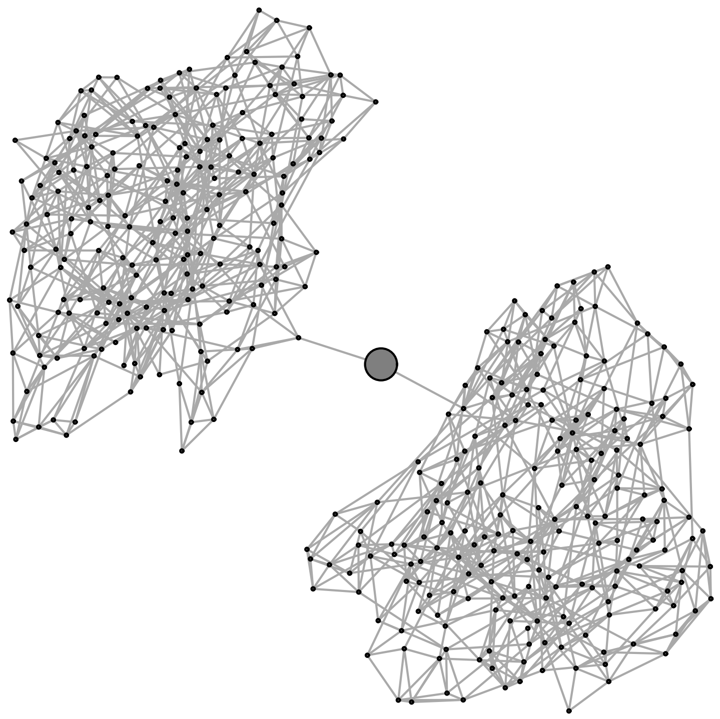


Figure 3.4: The large node in the graph has the highest betweenness, removing it would disconnect the graph into two separate subgraphs.

where n^i_{st} is the number of geodesic paths from vertices s to t which pass through vertex i, and g_{st} is the total number of geodesic paths from s to t. By convention it is usual to set $\frac{n^i_{st}}{g_{st}} = 0$ if there are no paths from s to t [74].

3.2.4 Closeness Centrality

Closeness centrality measures the importance of a vertex in the network by measuring its mean distance to other vertices in the network [74]. The logic behind this measure being that other vertices may need to be close to important vertices in a network above being close to unimportant vertices. For example, in a collaboration network of actors in which two actors are linked if they worked on the same film, an actor with high closeness centrality has likely worked with a large number of actors or a small number of actors who have worked with many others [74, 97]. Closeness centrality is defined as [74]

$$C_i = \frac{n}{\sum_j d_{ij}}, \tag{3.6}$$

where d_{ij} is the geodesic path length between i and j.

There are some problems with closeness centrality, for example it tends to have a low dynamic range due to a property of many networks in which the average geodesic path length between most nodes is small. It is not clear how to apply the measure when some vertices are completely disconnected from others, and is typically only used for groups of connected vertices.

3.2.5 Other Centrality Measures

There are many other ways to define the centrality or importance of a vertex in a network, and indeed there have been many centrality measures proposed in the literature. However, most of them do not find an immediate application in this thesis, but a brief overview of some of them will be given here for cohesion.

Perhaps the most obvious way to describe the importance of a vertex is by the number of connections to other nodes it has. This measure of centrality is called *degree centrality.* The degree is a fairly straightforward and intuitive way of measuring importance. For example, a person in a social network with many friends might be expected to be important. Although, this measure does not capture other important information. For example, the large vertex in Fig. 3.4 would have a low degree centrality but should probably be considered important in some way.

Another measure of centrality might consider the importance of a vertex's neighbours. For example, a person in a social network with a few very influential friends may be considered more important than a person in a social network with a large number of less influential friends. Quite a few centrality measures take this line of reasoning, with the most basic possibly being *eigenvector centrality* [12]. Eigenvector centrality works by first assigning an initial centrality estimate, $x_i = 1$, to all vertices. A better estimate x' is generated iteratively as

$$x'_i = \sum_j A_{ij} x_j, \tag{3.7}$$

where A is the adjacency matrix. In matrix form, $x' = Ax$, where x is the vector of elements x_i. After t steps a vector $x(t)$ of centralities is given by

$$x(t) = A' x(0), \tag{3.8}$$

The important property to be noted is that the eigenvector centrality measure will assign vertices with high degree a high importance as well as vertices which are connected to other important vertices. Eigenvector centrality has some problems on directed graphs that are addressed by other proposed centrality measures such as *Katz centrality* [46], *PageRank* [76], and *hyperlink-induced topic search* (HITS) [49]. However, directed networks are outside the scope of this thesis and these centrality measures will not be discussed here.

3.3 Properties of Overall Graph Structure

3.3.1 Clustering Coefficient

The number of triangles in many real-world networks is much higher than expected when compared to a network comprised of random connections, where the probability of finding a triangle should decrease with the size of the network. To illustrate the point: given a randomly connected network the degree of each vertex k would be roughly the same, based on the number of edges we chose to add to our random network. Given three vertices i, j, z, and knowledge that $\exists(i, j), (j, z) \in E$ then then it is straightforward to see that the probability of z being connected to i should be[1] $\frac{k}{n}$. Obviously as $n \to \infty$ the probability of $(z, i) \in E$ should become very small. As it turns out, this is not the case for a great number of real-world networks (which will be discussed a little later in this chapter, or see [68]).

It is easy to imagine why some real-world networks may contain so many triangles, consider for example the "a friend of a friend is my friend" relationship in social networks – given two friends, A and B, and a third person C, who is a friend to A, B is probably also a friend of A. Besides the dynamics of social relationships, in other networks it is easy to imagine that triangles may imply functional relationships or arise due to spacial proximity of vertices.

The *global clustering coefficient*, or simply *clustering coefficient*, or sometimes *transitivity*, provides a measure of the number of triangles in a network, normalized by the number of "possible triangles". A "possible triangle" or connected triple [68] is a set of three vertices connected by at least two edges such that if a third exists then the connected triple would also be a triangle. Therefore the clustering coefficient can be defined as

$$C = \frac{\textit{number of triangles} \times 3}{\textit{number of connected triples}}. \tag{3.9}$$

The factor of 3 exists because in an undirected network a triangle is counted three times depending on the vertex used as the starting point. This definition assumes triangles are being counted only once at each vertex, but if triangles are being counted twice (once for each direction) then the factor of 3 would become a factor of 6.

The clustering coefficient also has a local counterpart, C_i, which can be used to assign a clustering coefficient to a single vertex i. This is done by considering all pairs of vertices which are neighbours to i and computing the fraction of those neighbours which are also connected to each other. The *local clustering coefficient* is defined as [74]

$$C = \frac{\textit{number of pairs of neighbours of } i \textit{ that are connected} \times 3}{\textit{number of pairs of neighbours of } i}. \tag{3.10}$$

Note that the number of pairs of neighbours of i is $\binom{k_i}{2} = \frac{1}{2}k_i(k_i - 1)$. The local clustering coefficient can be considered as a measure of the number of redundant paths between neighbouring vertices, and is often found to have a lose dependence on vertex degree as vertices with higher degree tend to have lower values of C_i (although this is a lose observation, not a rule) [74].

[1]This is a fairly crude calculation but it suits the illustration and is not incorrect given the assumptions.

The average of the local clustering coefficients is also sometimes used as an analogue to the value of the global clustering coefficient, defined as [74, 97]

$$C_{WS} = \frac{1}{n}\sum_{i=1}^{n} C_i. \tag{3.11}$$

However, (3.11) is not equivalent to (3.9) and tends to be dominated by large numbers of nodes with small degrees, skewing the value away from what would be given by (3.9) [74]. In the context of this thesis the terms *clustering coefficient*, and *transitivity* will refer to the *global clustering coefficient* defined in (3.9).

3.3.2 Small-world Effect

The *small-world effect* states that the average distance between any pair of nodes in a network tends to grow only logarithmically with respect to the number of nodes in that network [74]. The small world effect does not apply to all networks, but it has been shown to be true in many [5, 97]. The property was demonstrated, and made famous by Stanley Milgram, a social scientist, in the 1960s [58] by his small-world experiments. Milgram demonstrated the small-world effect by sending 96 packages to randomly chosen recipients which instructed those recipients to attempt to get those packages to a specified target individual who lived over 1500 kilometres away. The initial recipients were given the name and address of the target but were instructed to send their packages only to people they knew on a first-name basis, who were then to do the same. Eighteen of the 96 packages arrived at their targets, and the mean length of the journey of the packages was found to be 5.9 steps. A remarkable result. This result may be surprising because [96]:

- The number of people in the world is large (billions).
- People are acquainted with a number of people, which at most is on the order of thousands.
- The network of acquaintances is decentralized, that is to say there is no one person who knows *everyone*.
- Circles of friends are usually overlapping – friends of one person are generally friends with each other.

However surprising, the small-world property explains many network phenomena such as why many message-passing systems are able to work efficiently on many kinds of decentralized networks. For example, communication packets sent across the Internet are typically expected to arrive in a small number of steps even though the exact location of the destination computer is not usually known by the sending computer. Examples of the small-world effect in real networks will be discussed further in Section 4.2. It is also important to note that the small-world effect is often used to refer to not only short path lengths, but a high clustering coefficient [97], as is observed in social and other networks.

3.3.3 Degree Distribution

The *degree distribution* refers to the probability distribution that corresponds to choosing a random vertex with a given degree within a network [74]. The degree distribution is constructed from a simple network by finding all of the values p_k for the number of nodes f_k with degree $k = 0, 1, ..., n-1$ such that

$$p_0 = \frac{f_0}{n}, p_1 = \frac{f_1}{n}, ..., p_{n-1} = \frac{f_{n-1}}{n}, \tag{3.12}$$

where n is the total number of vertices in the network. For networks which are not simple, the term $n-1$ would be changed to the maximum degree in the network. An essentially equivalent construct is the *degree sequence* of the graph, which is the sequence $k_0, k_1, ..., k_n$ where k_i is the degree of vertex i. Degree sequences are often sorted so that the degrees are listed in increasing or decreasing order.

The degree distribution can reveal quite a bit about a graph's structure. For example, Fig. 3.5 shows the (truncated) degree distribution of the network of autonomous systems on the Internet [70]. The skewness of the distribution shows us that the internet is dominated by systems with very few connections ($\sim$ 90% of all systems have fewer than 5 links), while a very small number of systems have a large number of connections (the largest degree is 2390 in this data set). The highly connected nodes are called *hubs*, and they are a common and interesting property in many real-world networks and will be discussed further in the next subsection. The degree distribution is often enough information to identify networks with very different underlying structure, or networks that were generated by very different processes [68, 74]. Consider Figure 3.7 that plots the degree distributions of two 100 vertex graphs, each one produced by a different graph model. One distribution shows that most vertices in the graph have a degrees of roughly 6, give or take 3 vertices, a relatively small range. The other distribution possesses hub nodes similar to those found in Fig. 3.5. From the degree distribution plots we can infer that the graphs are probably not similar in structure and this is typical of graphs produced by very different models.

Although the degree distribution can reveal interesting structural properties of networks, it should be stressed that it does not give all structural information alone (this is probably obvious). Consider, the two graphs in Figure 3.6; the graph on the left and the graph on the right have the same degree distribution but are not equivalent.

Scale-free Networks

Scale-free is a term that refers to networks with a particular kind of degree distribution, and in fact we have already seen a scale-free distribution in Fig. 3.5, and discussed some scale-free networks in terms of Price's cumulative advantage property found in citation networks. The term means that the degree distribution, at least roughly speaking, follows a power-law such at the probability p_k of a vertex having degree k is

$$p_k = Ck^{-\alpha}, \tag{3.13}$$

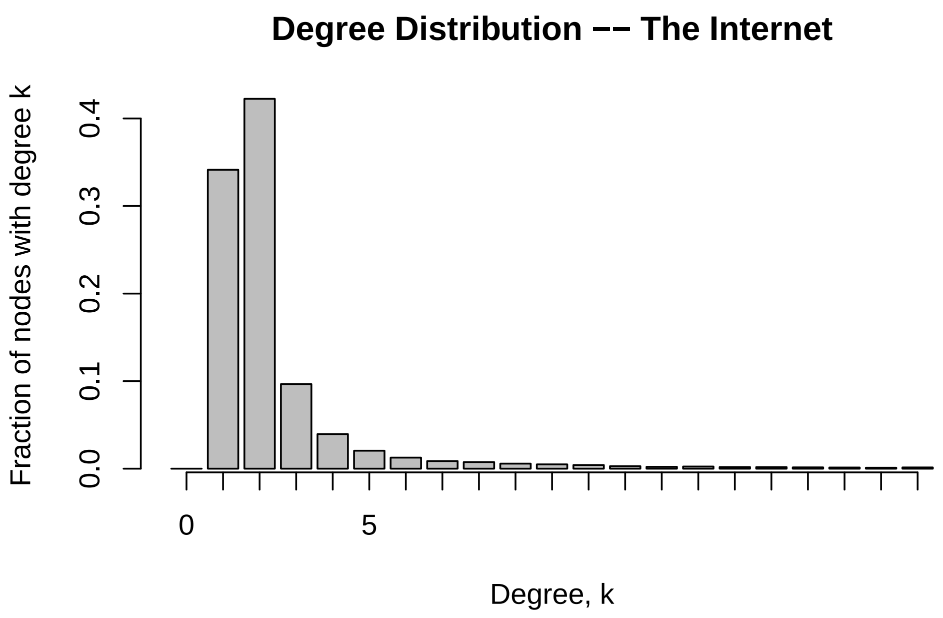


Figure 3.5: The degree distribution of the internet at the autonomous system level [70]

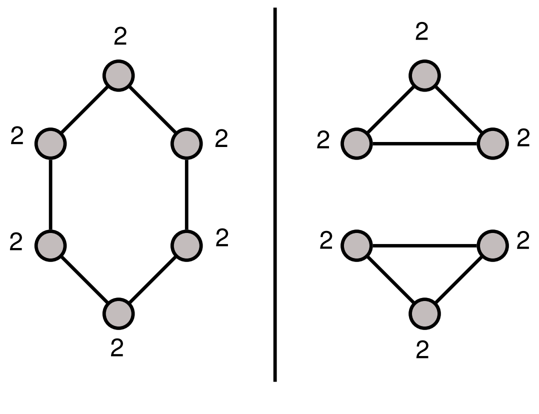


Figure 3.6: Two non-equivalent graphs with the same degree distribution

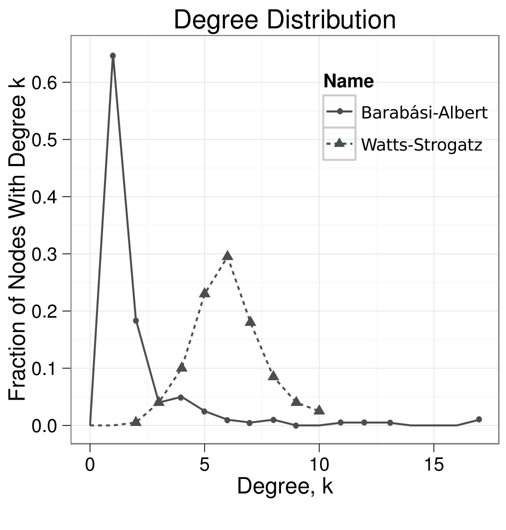


Figure 3.7: The degree distributions of a graphs produced by two different graph models.

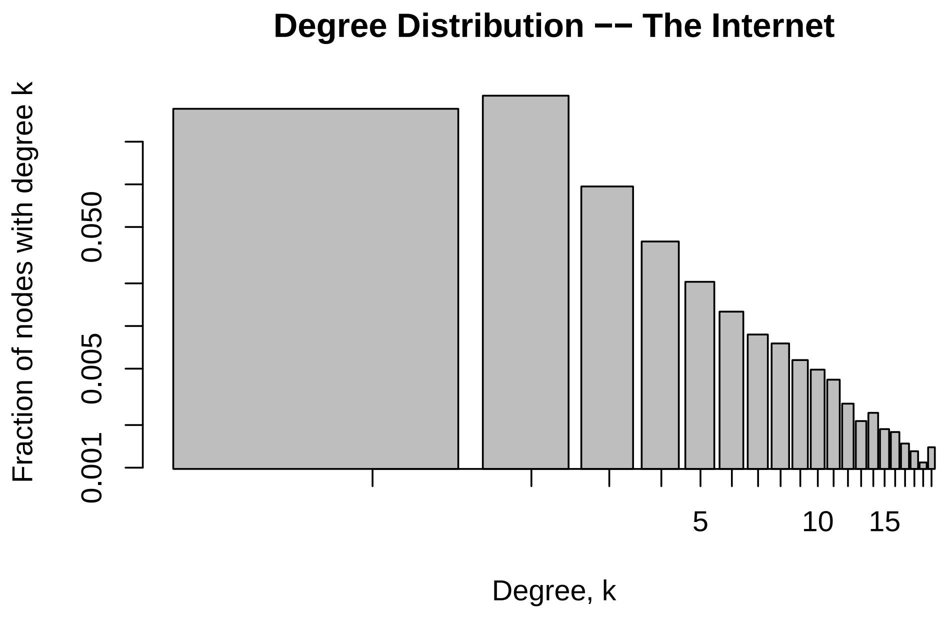


Figure 3.8: The degree distribution of the internet at the autonomous system level, logarithmic axes [70]

where C is some constant and, α is the exponent of the power law distribution. Many real world networks have a values in the range $2 \leq \alpha \leq 3$ [3,74]. If (3.13) is restated in terms of the logarithm of the degree distribution it is also plain to see that if the distribution follows a power law then vertex degrees should decrease linearly such that

$$ln(p_k) = -\alpha ln(k + c), \tag{3.14}$$

where $c = ln(C)$. If the distribution in 3.5 is re-plotted on logarithmic axes (a log-log plot) the relation is somewhat apparent as shown in Fig. 3.8, although the histogram is noisy because the bins appear to become smaller as the space along the x axis is compressed. A better option for plotting scale-free degree distributions is to adjust the bin sizes of the histogram so that they increase logarithmically, such that the n^{th} bin covers the interval $a^{n-1} \leq k \leq a^n$. The value of a is often chosen to be 2 [74]. In many real-world networks only the tail of the distribution follows a power-law, and in these cases excluding the first bin from the histogram will provide a cleaner looking plot. Figure 3.9 shows another log-log plot of the degree distribution of the internet at the autonomous systems level, this time with logarithmic bins with $a = 2$ and excluding the first bin. It is more obvious from this plot that the frequency of nodes of degrees decreases linearly as the degree increases, suggesting that at least the tail of this degree distribution probably follows a power-law.

Although plotting the degree distributions can provide some idea as to whether or not the degree distribution for the network in question follows a power-law, it can

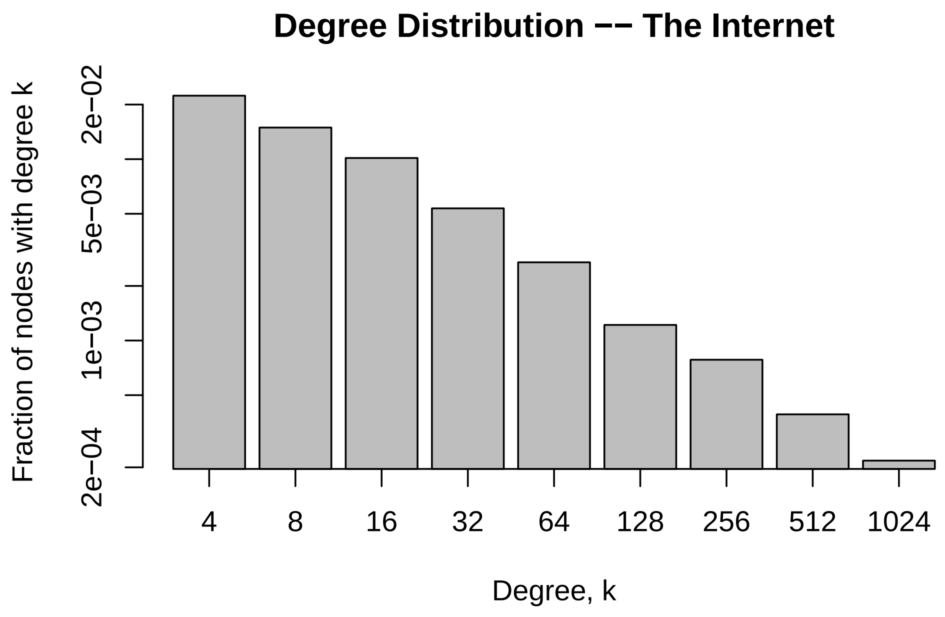


Figure 3.9: The degree distribution of the internet at the autonomous system level [70], log axes, log bin sizes. The linearly decreasing degree frequencies are a good indication that this is a scale-free network.

introduce bias. It is better to compute the value α by using the following formula [14]

$$\alpha = 1 + N \left[\sum_i ln \frac{k_i}{k_{min} - \frac{1}{2}} \right]^{-1}, \tag{3.15}$$

where k_{min} is the minimum degree for which the power law holds, N is the total number of vertices having a degree $k \geq k_{min}$, and the sum is over those same vertices rather than all vertices. The statistical error, σ, can be computed as [14]

$$\sigma = \sqrt{N} \left[\sum_i ln \frac{k_i}{k_{min} - \frac{1}{2}} \right]^{-1} = \frac{\alpha - 1}{\sqrt{N}}. \tag{3.16}$$

It should also be noted that (3.15) is an approximation of the full formula also provided in [14]. The full formula has no closed form, making it difficult to compute. The full form is required if k_{min} is less than about 6, however, the approximation is suitable for most networks [74]. The value of α, when computed for the autonomous systems data is about 2.1, falling within the expected range of power-laws for real-world networks.

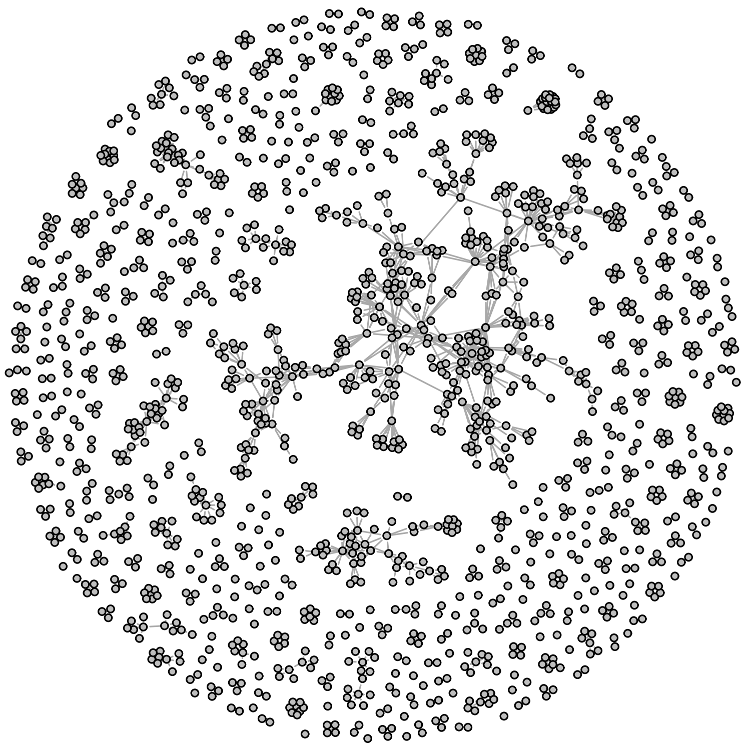


Figure 3.10: The co-authorship relations between scientists in the field of network science [69]. The network has many components, with one large one.

3.3.4 Components

A *connected component*, or simply a *component* in the case of undirected networks refers to a set of vertices that are interconnected such that every vertex pair in the component is connected by at least one path. Although they will not be discussed in detail here, directed networks require a slightly different definition of component. Directed networks have both *weakly connected components* in which every vertex pair is connected by a path if the edges are made unidirectional, and *strongly connected components* in which no modification to edges is needed to connect every vertex pair by a path.

Real-world networks are often made up of multiple components [74]. For example, a network of co-authorship patterns in the field of network science [69] as shown in Fig. 3.10 has one large component and many smaller ones. As it turns out, this is common of many networks and it would be unusual to find a network with two large components of the same size [74]. The reason for the ubiquity of one much larger component can be demonstrated by an intuitive argument. Suppose there were a network with two large components such that each contained half of the vertices. This would mean that there would be $\frac{1}{4}n^2$ possible pairings of vertices in different components, and if any of those pairs were to join then the two large components would become a single component. Similarly, suppose there is already a much larger component which includes $\frac{2}{3}n$ vertices and a smaller component which includes $\frac{1}{3}n$ vertices. There are $\frac{2}{9}n^2$ ways for those components to join.

The existence of components in a network can have some impact on other properties discussed in this chapter, such as the average geodesic path length. When two vertices are in separate components there is no path to join them, and so the geodesic path length between them is undefined. There are different approaches which can be used to handle this type of situation. For example, never consider two vertices which are not connected by at least one path, so when computing the average geodesic path the average of geodesic paths in each component is returned. Another approach is to assign the value of n which is greater than the maximum possible path length to the geodesic path length of node pairs in separate components. Similar complications arise when computing diameter, betweenness and closeness centrality. However as they all depend on computing geodesic paths these are handled implicitly by correcting the problem with geodesic path length.

3.4 Community Structure

The final property of networks which will be discussed in this chapter is called *community structure*. The term community structure comes from the study of social networks, but used generally it refers to densely connected groups of vertices with sparse connections between groups [71]. The densely connected groups are sometimes referred to as *modules*, *communities*, or *clusters*. Dividing a graph into separate communities based on natural divisions is called *community detection* or sometimes *clustering*[2]

Communities are different than components because more than one community can exist within a component. Dividing a network into communities is useful because it reveals additional organizational dynamics of the network beyond a single vertex or necessarily a fixed group of vertices. For example, recall the network representing associations between faculty members in a university shown in Fig. 2.1, the network is comprised of one large component, and portions of the network corresponding to departments are clearly more dense; a good community detection algorithm should be able to find them. Identifying parts of a network which are more densely connected by visual inspection can be a useful tool. However, visual inspection can introduce bias, and for very large networks it is not always possible. There have been numerous community detection algorithms proposed in the literature, and a good overview of methods brought together from many disciplines can be found in [25]. Only two methods will be discussed here to facilitate a general understanding of the problem because community detection is largely outside the scope of this thesis.

3.4.1 Modularity Optimization

A problem central to finding good divisions of communities within a network is to somehow define what is a "good" or "bad" community. One of the most intuitive, and therefore most common way to define a good community is by the *modularity*

[2]As opposed to *graph partitioning*, a related problem, which requires the number or size of the divisions to be known in advance. For a simple but effective graph partitioning algorithm see [47].

function or *quality function* [25, 73]. The modularity function compares the density of edges between groups of nodes to the expected density of a null model with the same degree sequence of the given network. If two nodes with degree k_i and k_j are connected when it is expected that they should only be connected with low probability then the connection is likely to be significant. The modularity Q of the network is defined as [73]

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(C_i, C_j), \tag{3.17}$$

where A_{ij} is the adjacency matrix, k_i and k_j are the degrees of vertex i and j respectively, m is the number of edges, and C_i and C_j are the clusters of i and j respectively. The function δ is the Kronecker delta which returns 1 if $C_i = C_j$, such that only nodes within the same cluster contribute to the sum. Note that the quality function only tells us how good a community assignment is, not how to assign communities.

Now that a quality function is defined, the modularity of the network can be optimised by any number of techniques, such as genetic algorithms, simulated annealing, or greedy algorithms [74]. Some more exotic methods are also possible using the modularity function, such as spectral-based community detection, which exploits properties of the eigenvectors of the adjacency matrix [71]. Figures 3.11 and 3.12 show the results of the leading eigenvector community detection algorithm [71] and a graph spectra based algorithm [17, 19] run on the immunoglobulin protein interaction network [34].

3.4.2 Betweenness Methods

Another way to find a community in a network is to identify those edges which straddle the boundary between two communities, called *between group edges* [74]. A common way to do this is to define an *edge betweenness* value which is an intuitive extension of the betweenness centrality measure discussed in Section 3.2.3. The edge betweenness value for an edge counts the number of geodesic paths passing through that edge [73].

A community detection algorithm can be defined then as follows. All edge betweenness scores are calculated, and the edge with the highest score is removed. This process is repeated, eventually the network will be split into many pieces and it will continue to be split until only single vertices are left. If the number and boundaries of the splits are kept track of, the algorithm can produce a tree-structure showing a hierarchical organization of the network which may be more useful than the clustered view generated by modularity optimization. Alternatively, a user could select one of many possible divisions which best suits their purpose. Figure 3.13 shows the dendrogram of hierarchical communities in the karate club network [100] as found by the edge betweenness community detection algorithm. Beside the dendrogram, Fig. 3.14 shows one particular division which had the most optimal modularity of the possible divisions.

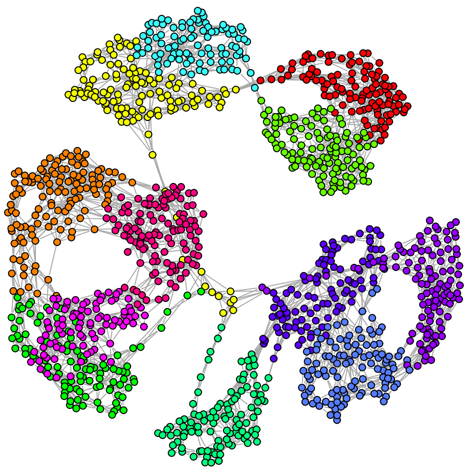


Figure 3.11: The community structure of interactions within the immunoglobulin protein [34] identified by the leading eigenvector algorithm [71].

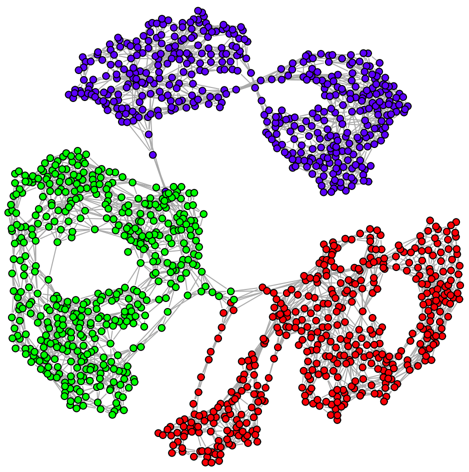


Figure 3.12: The community structure of interactions within the immunoglobulin protein [34] identified by a graph spectra algorithm [17, 19].

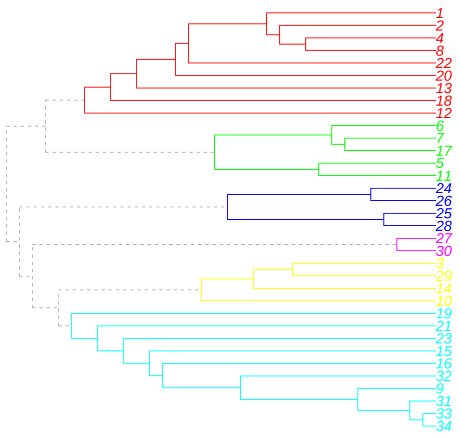


Figure 3.13: The dendrogram of the karate club network. The colours correspond to the divisions with the best modularity.

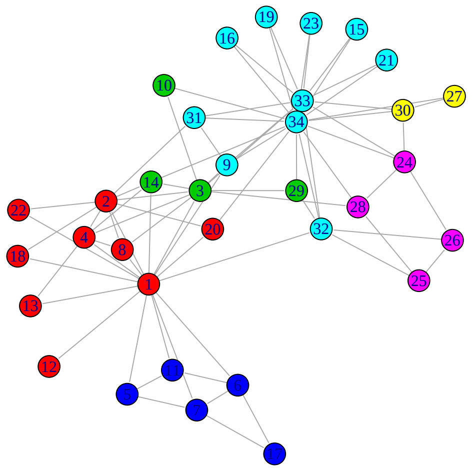


Figure 3.14: One particular division of the karate club network.

Chapter 4

Graph Models

Graph models are a valuable tool for the study of complex networks because they allow the characterization of network organization without having to describe the entire network. For example, a graph model could be created to characterize the way business collaborators interact, and then be used to make predictions about their interaction or simulate scenarios that may influence their actions. The graph model could be applied again and again for any group of business collaborators (probably within some constraints, e.g., people who own small businesses may socialize differently than people who own large ones). The business collaborators example highlights two things of note. First, that graph models allow us to generalize on observations of real networks so that we may characterize other similar ones. Secondly, once a graph model is designed it may only be robust to certain kinds of networks. However, despite the differences between many real-world networks, scale-free degree distributions and the small-world effect appear to do a good job of characterizing a large number of real-world networks.

This chapter introduces the Erdös and Rényi (ER) random graph, the Watts-Strogatz (WS) small-world model, and the The Barabási-Albert (BA) model, followed by a few other common models. The models presented here were chosen for their historical significance and impact on the study of complex networks and graph models, and are later used in this thesis to generate benchmark data. The ER model is perhaps the first graph model ever studied, while the WS and BA models are important because they showed that the small-world effect and scale-free degree distributions could be used to characterize a large number of real networks. The graph models discussed in this chapter are by no means an exhaustive list. Many graph models exist and more extensive surveys can be found in [37, 74].

4.1 The Random Graph Model

The Random graph model is usually associated with Erdös and Rényi and their 1959 publication [23], although earlier work on the random graph appeared in 1951 by Solomonoff and Rapoport [87]. It is also sometimes called the Poisson random graph, or the Bernoulli random graph. This model generates a graph by adding n vertices, and adding each possible edge with probability, p. This produces a graph denoted $G_{n,p}$. The model represents the *ensemble* of all $G_{n,p}$ graphs in which a graph having m edges appears with probability $p^m(1-p)^{(M-m)}$, where $M = \frac{1}{2}n(n-1)$ is the maximum number of edges [68]. In the limit of large n, keeping the mean degree $z = p(n-1)$

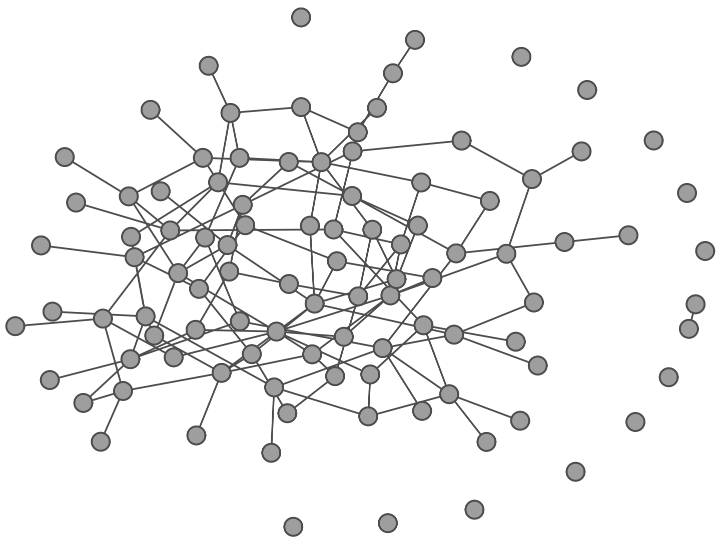


Figure 4.1: A graph generated by the Erdös and Rényi model with $n = 100$, and $p = 0.022$

constant, the probability, p_k, of a vertex having degree k is

$$p_k = \left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k} \simeq \frac{z^k e^{-z}}{k!} \tag{4.1}$$

with the approximation becoming exact in the limit of large n and fixed k. Thus, the degree distribution for the random graph is a Poisson distribution [68]. Figure 4.1 is an example of a graph generated by the Erdös and Rényi model. Note that the graph is not necessarily connected, and that there is one component that most of the vertices belong to. The average number of edges $\langle m \rangle$ for some p can also be calculated as

$$\langle m \rangle = \left(\begin{array}{c} n \\ 2 \end{array} \right) p, \tag{4.2}$$

which is simply a statement that the expected number of edges is the expected number of edges p, between a pair of vertices, multiplied by the number of pairs of vertices in the graph. Given (4.2) and that the mean degree $\langle k \rangle$ of a vertex in an undirected graph with m edges is $\langle k \rangle = 2m/n$, it is straightforward to see the mean degree of a vertex in a random graph is

$$\langle k \rangle = \frac{2}{n} \left(\begin{array}{c} n \\ 2 \end{array} \right) p = (n-1)p. \tag{4.3}$$

While the random graph was an important first step in graph modelling it is now understood to be a poor model of complex networks [74]. The Poisson distribution of node degrees is generally unlike real networks, and by definition the links between elements in a complex network are not entirely random. However, the random graph continues serves as a useful null model by which to detect non-random organization.

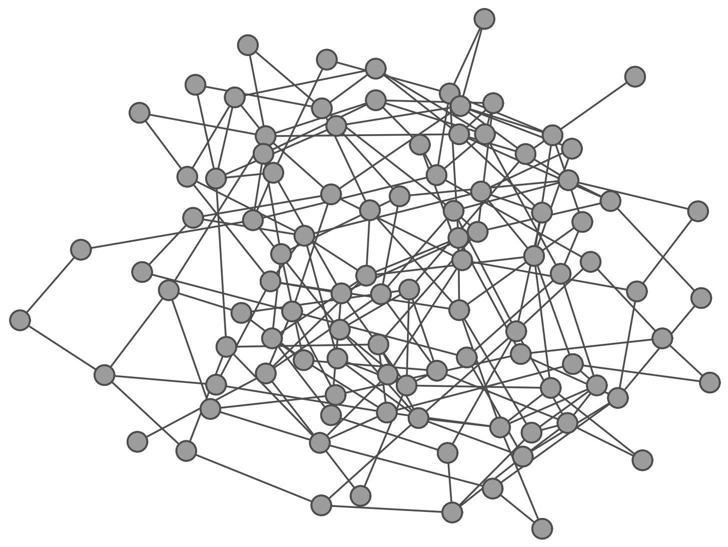


Figure 4.2: A graph generated by the Watts-Strogatz model where $n = 100$, and $p_{rw} = 0.2$.

For example, social networks tend to have clustering coefficients orders of magnitude higher than what is found in random graphs [3,72,74,97].

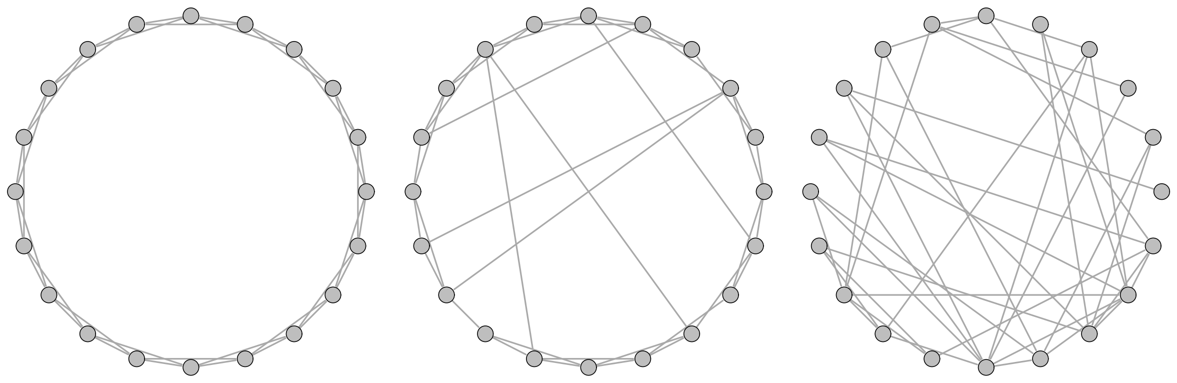
In fact, many real-world networks have clustering coefficients much larger than random graphs (see Section 3.3.1 for an explanation why triangles are unexpected). The difference between random networks and real ones is so stark with respect to triangles that it is expected that the triangles must possess a function, one of the primary observations leading to the creation of the small-world model.

4.2 The Small-world Model

The Watts-Strogatz (WS) small-world model focuses on modelling two properties observed in real-world networks. First, there tends to be a large number of triangles in many real-world networks [97], the small-world model can reproduce this. Secondly it maintains a small average geodesic path length between any two vertices as network size grows large, the so-called *small-world effect* [74,97].

The Watts and Strogatz *small-world* model builds graphs by creating a ring of n vertices in which each vertex is connected to each of its neighbours up to k spaces away from itself. Each edge is then considered in turn and with probability p_{rw} one of its ends is “rewired” to a new vertex chosen uniformly at random. This results in a high connectivity among a vertex’s neighbouring vertices as well as a low upper bound on average geodesic path length. Figure 4.2 shows a graph generated using the Watts-Strogatz small-world model. Figure 4.3 illustrates how shortcuts are used to decrease the average geodesic path length and how the graph approaches a random one as p_{rw} increases, thereby decreasing the number of triangles.

While the Watts-Strogatz model produces graphs with transitivity values and path lengths that are similar to some real-world networks, it fails to produce graphs



Rewire probability $p_{rw} = 0.0$, each node has 4 neighbours – lots of triangles, but path lengths are long.

p_{rw} is increased to 0.1 – fewer triangles, but path lengths are shorter.

p_{rw} is increased to 0.5. The graph approaches random.

Figure 4.3: The relation between average geodesic path and transitivity in graphs generated by the WS model, $n = 20$.

with realistic degree distributions. It can be shown that the degree distribution of small-world graphs corresponds to [74]:

$$p_k = e^{-cp} \frac{(cp)^{k-c}}{(k-c)!} \tag{4.4}$$

where p_k is the probability of a vertex having degree k, c is the initial degree of each vertex before the rewiring process, p is the probability of rewiring. However, despite this shortcoming the small-world model has been used to characterize a large number of real-world networks. Including power grids, airline routes, the collaboration network of movie actors, high school friendships, the neural network of the *C. elegans* worm, and more [5].

Depending on the application, the unrealistic degree distribution may or may not be a problem. The model was never intended to produce graphs with distributions that matched those found in real-world networks – the transitivity and path length were the main considerations. Models such as the Barabási-Albert model explicitly provide mechanisms for reproducing the degree distributions found in many real-world networks.

4.3 The Barabási-Albert Model

Many real-world networks such as the WWW, social networks, the nervous system, the interactions of proteins and more follow a power law degree distribution [7, 68]. The Barabási-Albert model [7] provides a mechanism to create networks with power-law degree distributions, related to the cumulative advantage process observed by Price [79]. In such networks, the probability $P_k \approx k^{-\alpha}$, where α is some constant

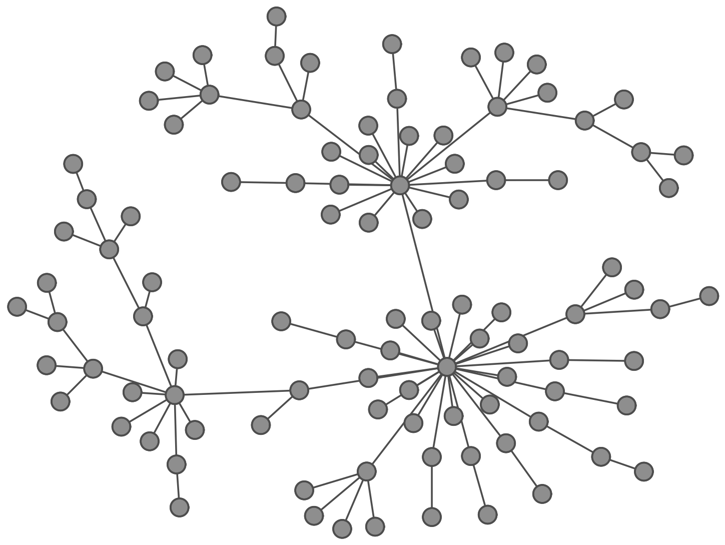


Figure 4.4: A graph generated by the Barabási-Albert model with $n = 100$, and $m_0 = 1$

(typically in the range $[2, 3]$), is the probability that a vertex chosen uniformly at random has degree k.

The Barabási-Albert (BA) model was premised on two important observations. Firstly, that real-world networks grow over their lifetime. Secondly, the higher the degree of a vertex, the greater is its propensity to collect new edges, this is known as *Preferential attachment* [3, 7, 68]. Two mechanisms were proposed to describe the behaviour, and they work as follows:

1. *Growth:* Starting with (m_0) vertices, at each time step add a vertex with, $m \leq m_0$ edges, and attach to m different existing vertices.

2. *Preferential attachment:* Connect the new vertex to vertex i with a probability dependent on the degree k_i of i, such that

$$\Pi(k_i) = k_i \left[\sum_j k_j \right]^{-1}. \tag{4.5}$$

It can be shown that after t time steps the scale-free network will have $n = t + m_0$ vertices and mt edges. Figure 4.4 shows a graph generated using the Barabási-Albert model.

The BA model is not able to generate all kinds of power-law degree distributions, and in the original study it was shown that scale-free behaviour was only present for linear preferential attachment, or $\alpha = 1$ [7]. The model, typically generates graphs with a tree (or near tree-like) structure, and so naturally such graphs have a clustering coefficient of zero. Although, an increase in the value of m can increase the clustering coefficient, it does so at the expense of creating a more dense network (the minimum node degree must be m), and it does not guarantee better transitivity in the limit of large graphs [50]. The limitations also mean that even if the BA model is capable of reproducing the clustering coefficient found in a real-world network, it may not be able to model it with a high degree of accuracy without impacting the goodness of fit of the degree distribution.

4.4 Other Graph Models

This section will provide an overview of some other less-common graph models which have are of some interest with respect to this work, although they may not be mentioned explicitly in later chapters. The Forest Fire (FF) model, Klemm's model and the Duplicate-Diverge (DD) model all propose additional mechanisms for constructing scale-free behaviour although they create different networks in some respect than the BA model.

4.4.1 Forest Fire Model

The Forest Fire model [55] was proposed to address two points which were not addressed by previous models. First, that the average vertex degree increases over time in many networks, so that the network becomes more dense (in fact, they find that the number of edges increases exponentially with the linear increase of vertices). Second, that the diameter of many networks also shrinks in accordance with the increase in node degree. The observations made appear to hold for a number of real-world graphs, including citation networks, collaboration networks, and the autonomous systems level of the Internet [55]. The authors proposed a model based on the spread of forest fires to address their observations. The FF model probabilistically creates outgoing links with a *forward burning* probability and incoming links with a *backward burning* probability, to create directed graphs. The method can easily be generalized to undirected graphs by making all links bidirectional after they have been established. The FF model is described as follows:

1. Choose a forward burning probability p and a backward burning probability r.

2. Consider a node v joining the network at time $t > 1$, G_t is the graph so far.

3. v chooses an ambassador node w uniformly at random and connects to it.

4. Two random numbers, x and y, are chosen from a geometric distribution with means $(1-p)^{-1}$ and $(1-rp)^{-1}$ respectively.

5. v selects x out-links and y in-links incident to nodes not yet visited. Let $w_1, w_2, ..., w_x + y$ denote the other ends of the selected links. If not enough in or out-links are available, v selects as many is it can.

6. v forms out-links to $w_1, w_2, ..., w_x$ and then applies step 4 and 5 recursively to each of $w_1, w_2, ..., w_x$. As the process continues, nodes cannot be visited a second time, preventing the construction from cycling.

The FF model generates heavily tailed degree distributions, which are intended to mirror the evolution of real-world networks over time, and addresses some perceived problems with the BA model. It works by "burning" outward starting from w, though to w_x, proceeding recursively before dying out [55]. The model as presented here creates directed networks, although they could easily be converted to undirected networks after they are generated.

4.4.2 Duplicate-Diverge Model

The Duplicate-Diverge (DD) model [93] was proposed as a possible mechanism to describe the dynamics of interacting proteins, which was derived from observations of biological systems. The DD model works on two simple principles, duplicate and diverge. These describe how a vertex once added to the network copies some links from another vertex and then diverges from its behaviour by randomly rewiring some connections. The model is very simple, and is similar in many respects to other vertex copying models initially proposed for simulation of the structure of the web [48]. The duplicate and diverge steps are described as [93]:

Duplicate A vertex i is selected at random. A new node i', with a link to all the neighbours of i, is created. With probability p a link between i and i' is established.

Diverge For each of the nodes j linked to i and i', one of the links (i, j) or (i', j) is randomly chosen and removed it with probability q.

The DD model is able to create degree distributions in which the probability of a vertex gaining another link is proportional to $(1-q)\frac{k}{n}$, and it will generate triangles with a probability $(1-q)$ [93]. The model is interesting because it independently proposes a mechanism entirely different but equivalent to preferential attachment for scale-free network behaviour, premised on observations of biological systems. Furthermore, it is also interesting that the duplication mechanisms potentially responsible for scale-free behaviour in protein interaction networks have also been proposed as a model of the WWW.

4.4.3 Klemm-Eguílez Model

The last model which will be discussed in this section was proposed by Klemm and Eguílez [50] as a method for growing scale-free networks which also possessed small-world behaviour (including high clustering coefficients), unifying the two properties. The model is described as [50]:

- Create a fully connected network of size m.
- The first m nodes go into an 'active' list
- Introduce $n - m$ remaining nodes one at a time, attaching each to one of m existing nodes from the active list with probability $1 - \mu$.

  - If a node from the active list is not chosen, it is added to the active list after removing a node already in it with probability $p_i = ak_i^{-1}$, where $a = \sum_j k_j^{-1}$; j is the number of nodes in the active list.

The proposed model produces degree distributions such that a vertex has probability, $P(k) = 2m2k - 3, (k \geq m)$ of having degree k and is shown to have a maximum clustering coefficient value of $C = 5/6$ in the limit of large n [50]. The model has some similarities to the BA model (adding nodes iteratively, probability of attachment related to degree), but introduces a mechanism for the reliable increase of clustering coefficients even in the limit of large n.

Chapter 5

Genetic Programming

The previous chapters have introduced complex networks, and properties with which to describe and analyze them, as well as the concept of a graph model and some specific human-designed graph models. The question remains, how do we infer a model from the structure of a network? When humans are faced with the task of designing models they typically look for interesting properties within certain networks, such as scale-free degree distributions, or high clustering coefficients and low path lengths, and then attempt to define a mechanism for creating those things (as has been the case with the models in Chapter 4). This thesis proposes that the process of finding interesting structure within a network and designing a mechanism to express that structure can be automated, and furthermore, proposes that the automation can be done by way of genetic programming.

Genetic programming (GP) is an evolutionary paradigm modelled after Darwinian evolution in which populations of randomly generated programs compete, mate, and breed new programs [52]. The programs are selected probabilistically for mating proportionally to their fitness. This means fit programs tend to mate together, producing offspring which are usually more fit than their parents. Fitness is usually a user-defined function and describes how well a program achieves a particular goal. The search that GP performs tries to optimize the fitness of the individuals, so an appropriate fitness measure is crucial for effective evolution because it actually describes the objective of the search. An *inappropriate* fitness measure will therefore guide the search away from the intended region and produce undesirable results.

In the past, GP has proved successful in learning suitable programs to perform a wide variety of tasks such as classification, symbolic regression, growing various structures, and other inference problems such as bio-network modelling [77, 81]. It has also been shown to find good results in areas where the domain is not well understood [52–54], making it a natural fit for inferring graph models for complex networks. The general flow of execution of a GP is given in algorithm 1.

Algorithm 1 Genetic Programming

```
Randomly generate initial population;
while termination criteria is not met do
    Execute each program and assign a fitness;
    Select two programs probabilistically according to fitness;
    Create new individuals via crossover and mutation;
end while
Return best individual so far;
```

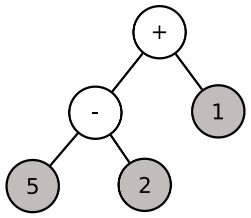


Figure 5.1: A GP Tree

5.1 GP Chromosomes

Programs, called *chromosomes*, in a GP population are constructed from a set of primitives called the *language*, or *function set*, which consists of operators that are relevant to manipulating data which corresponds to the problem to which the GP is being applied. The structure of the chromosome varies per GP system, but commonly they are represented as parse trees. Operators are placed in the tree according to their arity – operators which take arguments form the internal vertices and operators which return data and take no arguments, called *terminals*, form the leaf vertices. In order to translate the GP chromosome into a program the operators are executed on their operands, recursively, starting from the root vertex. For example, the GP tree shown in figure 5.1 would parse to the expression $(5 - 2) + 1$.

5.2 Population Initialization

In tree-style GP the initial population is generated by randomly building a tree. Trees can either be of the *full* style or of the *grow* style [77]. *Full* trees are constructed by selecting non-terminal vertices until the tree reaches a certain depth, at which point terminal vertices are selected to ensure a valid GP individual has been created. *Grow* trees are constructed by selecting either terminals or non-terminals according to some probability, and like the *full* trees a depth limit for selecting non-terminals is imposed to keep trees from becoming arbitrarily large.

Full trees tend to be very bushy, whereas *grow* trees are sparse. One or the other style of tree may be more suitable for a particular problem or search space. Some researchers have found it advantageous to use both full and grow methods to each generate half of the initial population generation in order to generate a wider variety of tree shapes, this is called *ramped half-half* [77].

5.3 Multi-Objective Fitness

Once a population has been generated, some indication of which chromosomes are "good" or "bad" is required so that suitable programs can be selected for breeding. This is accomplished by fitness. Fitness measures the suitability of an individual in the population for its intended purpose, and describes the objective of the search. For many problems there is often only one objective to optimize, e.g., generate a program which will complete a task quickly. In such a problem, time is the only consideration, so programs that are able to complete the task are automatically better than those that cannot, and those that can do it quickly are better than those that do it slowly, etc. However, for many other problems there is more than one objective of interest, e.g., complete two tasks quickly. In these cases it is not always clear how to assign a fitness, and a *multi-objective fitness* strategy is needed.

There have been many multi-objective fitness strategies proposed in the literature [15] and handling multiple objectives is non-trivial. Quite often the properties which are being optimized are not independent from each other and changes in one will impact the other, sometimes in ways which are at odds with one another. The simplest way to handle multiple objectives is to sum the values of each fitness measure together. However, if the fitness values from each objective are not on the same scale, then one objective could be under or over-represented in the sum. The obvious solution to the scaling problem is to add weights, so the fitness becomes a weighted sum of the fitness values of each objective. Weights now introduce a new problem, how do you decide on a suitable weight assignment? If the weights are used to scale the objective values so they contribute equally then a decision has been made that each objective is equally important, maybe this is not the case. It may be equally likely that a decision cannot truly be made about the importance of the objective (consider an example where both safety and cost of a solution are being optimized). Most proposed multi-objective strategies provide some solutions for the problems with a weighted sum fitness, although weighted sum continues to be useful for ease of implementation and in cases where weighting schemes are obvious. This thesis has made some use of the *summed-ranks*, and *Pareto* multi-objective strategies as well as weighted sum in conjunction with an adaptive weighting mechanism in Chapter 6, 7, and 8 so they will be defined here.

5.3.1 Summed-Ranks

Summed-ranks [10, 29] works by assigning each individual j a raw fitness value for each objective i. Individuals are then assigned a rank $Rank_{ji}$ with respect to each objective, such that zero is the worst rank. The fitness F_j of an individual then used for selection for a maximization problem is defined as:

$$F_j = \frac{\sum_{i \in O} w_i \times Rank_{ji}}{\max(Rank_i)} \tag{5.1}$$

where O is the set of objectives and w_i is the weight placed at objective i. This is the same formulation given by Bentley and Wakefield in [10] for their *Weighted Average Ranking* with the addition of a rank normalization by the maximum rank in each objective. Generally speaking, rank weightings are kept at 1 to preserve the independence of ranks, although more exotic weighting schemes are possible at the risk of re-introducing the problem of weight tuning.

Adaptive Weighting

To eliminate the need to search for useful objective weights, an adaptive weighting mechanism is devised here which can be used in conjunction with weighted normalized summed-ranks or weighted sum. The weight or rank weight for each objective $i \in O$ at generation t, denoted $w(t)_i$ was computed as:

$$w(t)_i = \frac{\sqrt{1.0 - \overline{A}(t-1)_i}}{\sum_{j \in O} \sqrt{1.0 - \overline{A}(t-1)_j}} \tag{5.2}$$

where O is the set of objectives, $\overline{A}(t-1)_i$ is the average adjusted fitness value (where fitness is normalized to $[0, 1]$, 1.0 being the best possible fitness) for objective i at generation $t-1$. In the case of weighted normalized summed-ranks the value for $w(t)_i$ is used in place of w_i in (5.1). See Section 6.2 for definitions of adjusted fitness values.

5.3.2 Pareto Ranking

Pareto ranking is a common strategy for handling multiple objectives with GP because it preserves the independence of objectives [36]. If we define an individual's fitness as a vector of objective fitness values, then we say individual x *dominates* individual y, when each element of its fitness vector, $\vec{x}$, is less than the corresponding element in y's fitness vector, $\vec{y}$ (assuming a minimization problem). That is to say, $\vec{x}$ *dominates* $\vec{y}$ *iff* $\forall i \in O : (x_i \leq y_i) \wedge \exists i \in O : (x_i < y_i)$. If Pareto ranking is used, an individual's Pareto rank is used for selection.

5.4 Selection

Selection is the mechanism which guides the search through the search space and provides the necessary competition for effective evolution, pitting programs against each other to compete for the opportunity to reproduce, and have their offspring move on to future generations. Selection works by probabilistically choosing two individuals from the population, based on their fitness values. The selected individuals are then mated and their offspring are placed in the next generation. The amount of importance the selection mechanism places on the fitness of the individual during the selection process is referred to as the *selection pressure*. Usually selection methods are tuned to exert a moderate amount of *selection pressure* on the search.

The two most common selection methods are *tournament* and *roulette* selection [77]. Roulette selection selects a program i with probability

$$p_i = \frac{f_i}{\sum_j^N f_j}, \tag{5.3}$$

where N is the size of the population. The probability of selection is proportionate to fitness, but it is straightforward to see that if all programs have similar fitness values (as is common toward the end of a GP run) it may be probable that the worst programs are selected for breeding. The ability of the selection mechanism to drive the search toward good solution is called *selection pressure*, and roulette selection is not always able to apply even selection pressure. However, it is possible to adjust selection pressure by applying various transforms to the distribution of probabilities p_i such that the selection pressure can be increased. A more straightforward and elegant solution is often to simply use tournament selection.

Given k individuals, *Tournament selection* chooses the best. The mechanism is easy to implement, and as a side effect of its operation tournament selection is able to apply equal selection pressure over the duration of the search, regardless of whether or not all members in the population are similar in fitness. The selection pressure can also be adjusted by increasing the size of the tournament. For its simplicity and ability to apply constant selection pressure tournament selection is often preferable to roulette selection, and will thus be the operator of choice in this thesis.

5.5 Genetic Operators

Once all the individuals in the population have been assigned a fitness value, and a pair of individuals are selected for reproduction, they are subjected to the genetic operations of *crossover* and *mutation.* Crossover is the act of mating two individuals such that their genetic material is combined to produce offspring. In tree-style GP crossover usually takes the form of *subtree crossover* where a crossover point is chosen randomly from both parents and the subtrees from those points are swapped. Figure 5.2 illustrates the result of a mating of two parent trees to form two offspring. The dark vertices indicate the crossover points.

While crossover is valuable as a exploratory strategy during search, it does not create any new genetic material in the search and it also tends to make fairly large movements through the search space. Additionally, it depends on adequate diversity among the population to produce useful offspring. For more subtle movements, and to introduce new genetic material into the population a mutation operator is used. Mutation makes small, random, changes to offspring before they are placed into the next generation and it is usually applied with some small probability. In tree-style GP mutation often takes the form of *subtree mutation* where a random subtree is generated and then inserted in the tree at some random point. Now that GP has been introduced, the following chapters focus on how GP was used to infer graph models of complex networks and the efficacy of the approach.

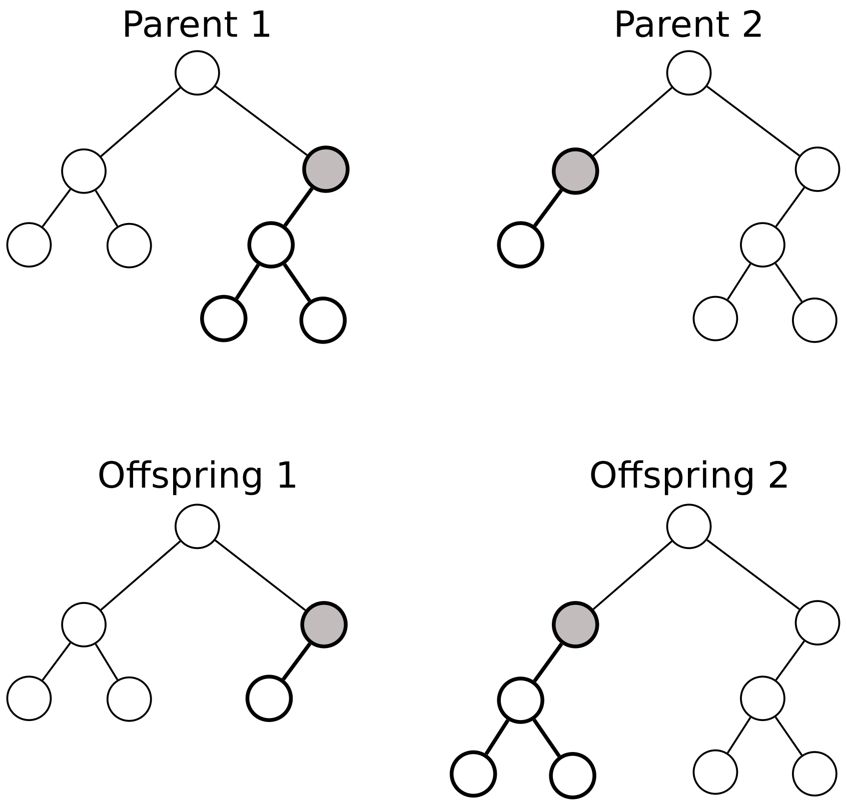


Figure 5.2: Subtree crossover

Chapter 6

Evolving Graph Models of Complex Networks

The GP system for automatically inferring graph models for complex networks was built using the *RobGP* system, a Koza-style GP system written in C++ [28, 29]. Individuals in the population are represented by program trees where functions create the internal nodes and the leaf nodes are so-called terminal values, which are made up of nullary functions or constants. The initial populations can be randomly generated using the "full" or "grow" operations and they can be combined to perform a "ramped half-and-half initialization" [52]. Program trees in the population are recombined during a crossover and mutation breeding phase using a subtree crossover that swaps a subtree from each parent, and grow mutation, which replaces a randomly selected subtree within a program with a randomly generated one. To chose individuals from the population for either of the breed phases tournament selection is used. Parameters used with the system are given with the results presented in Chapter 7 and 8. Network properties were computed using the *iGraph* C library [17]. All operations were strongly typed [61].

The system was designed in order to evolve a graph model given a specific network, called the *target network*. The evolved models should produce graphs that are similar to the target network with respect to the fitness measures. The following sections describe the challenges encountered in designing the system, fitness measures, the shape of the GP tree as well as the GP language used.

6.1 Challenges

When this work began, it was unclear whether or not it was even possible to accomplish the automatic inference of graph models. Genetic programming seemed a logical starting point because it is good at building structures and has been successful in the past at deriving solutions to problems when little is known about the domain. However, the choice to use GP necessitates the definition of at least one fitness objective, as well as the definition of a set of functions with which the GP can construct models. The GP function set will be discussed in Section 6.3.

Defining a fitness function is that it requires knowledge of some ideal target performance. In the case of evolving a graph model the ideal is to actually reproduce the process which created the network at hand. Of course, if the process that created the network was known, then there would be no reason to derive a graph model. So, if there is no knowledge of the ideal performance then how can performance be measured? The answer to this question is to use the available information embodied in

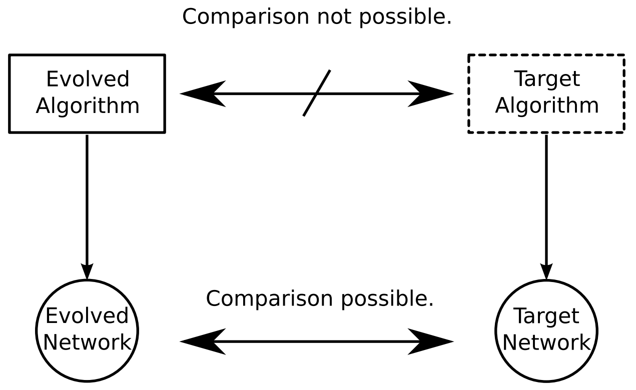


Figure 6.1: The comparison necessary to generate a fitness value.

the network for which a graph model needs to be derived, but obviously an algorithm cannot be compared to a network so the problem becomes a comparison of networks (see Fig. 6.1).

Comparing networks presents two immediate challenges, the first is that there is no known polynomial time algorithm for comparing graphs, meaning a structural comparison metric is prohibitively expensive with respect to computational time for any network of interest. The second challenge is a practical one. Real-world network data is often hard to come by, because the collection of connectivity data for large decentralized systems is a non-trivial task, and with GP it is typical to have a set of various training examples on which to test the evolved programs. Due to the uniqueness of many network data, it cannot be assumed that data for more than one network of the same category is available.

There is not much to be done about the unavailability of data. However, if one considers that a complex network is decentralized in nature and that the process of growing it comes from how each node connects itself, then perhaps the network should be considered as a collection of separate (each node behaves independently) but not independent points of data (we are interested in the emergent behaviour and organization of the group). In this view each node is itself a training example. Although, ultimately the features of interest only become apparent after a number of nodes is generated, so only after an evolved model generates approximations of all training examples do we measure performance. The solution to the challenge of designing the fitness functions will be discussed in the next section, and came from an examination of the methods used to categorize real-world networks, as well as the way in which the behaviour of diffusive processes have been estimated on epidemiological networks.

6.2 Selection and Fitness Assignment

Commonly, graph models designed by humans have focused on reproducing statistical features of real-world networks that differ from randomly connected networks, such as a large clustering coefficient, distinctive degree distributions, or small average path lengths [7,11,50,97]. Although an entire network cannot be fully described by a small collection of statistical measures, they are tractable to compute, are commonly used to classify different types of complex networks [74], and may be sufficient to describe sets of networks that share similar behaviour. In particular, there is empirical evidence to support that diffusive behaviour on a network can be predicted with a high degree of accuracy (e.g., in an epidemiological network) using the degree distribution, the clustering coefficient, and average path length [6,94].

Given that we wish to produce an algorithm that generates graphs that can describe real-world network behaviour, it is logical to produce algorithms that generate graphs that are similar to a given real-world network with respect to features that are known to affect such behaviour. The fitness functions are therefore designed to compare the target network to graphs produced by the evolved models, called the *active graphs* with respect to the degree distribution, average path length, and the clustering coefficient. It should be noted that while these three features are known to provide a fairly accurate general description of many real-world networks, it is not known if there is an optimal set of features to consider. Some properties may be more useful than others depending on the target network, and the method proposed here can easily be extended to include or exclude any desired property.

Each feature considered for comparison necessitates its own fitness objective, so this is a multi-objective problem. We consider five different methods for handling the multiple fitness objectives. These include a standard weighted sum, a normalized *summed-ranks* strategy [10], Pareto ranking [36] and an adaptive weighting scheme used in conjunction with summed-ranks and weighted sum, which were introduced in Section 5.3. The fitness objectives are described below. The raw fitness objectives are defined as:

Minimize:

$$F_1Raw = |l(G_t) - l(G_a)| \tag{6.1}$$

$$F_2Raw = [C(G_t) - C(G_a)]^2 \tag{6.2}$$

$$F_3Raw = D_{G_t,G_a} \tag{6.3}$$

$$F_4Raw = \frac{1}{|H|}\sum_i \frac{(h_i^t - h_i^a)}{h_i^t + h_i^a}. \tag{6.4}$$

Where $l(G_t)$ is the average geodesic path length of the target graph, $l(G_a)$ is the average geodesic path length of an active graph, $C(G_t)$ and $C(G_a)$ are global clustering coefficients of the target and an active graph respectively, and $D_{t,a}$ is the KS-test statistic comparing the degree distribution of the target graph to an active graph. $|H|$ is the number of discrete values in the degree distribution histogram, h_i^t and h_i^a are the i^{th} values in the target network and active graph's degree distributions respectively.

Note that there are two functions, (6.3), and (6.4), comparing the degree distributions of the active and target graphs. This is somewhat redundant, but Eq. (6.4) was introduced because (6.3) had a tendency to under-emphasize the importance of low-frequency high-degree nodes in a degree distribution (see Appendix A for details). Consider a star graph that contains 99 nodes with one edge and a single node with 99 edges. Eq. (6.3) would consider missing a one-degree node as importantly as missing the hub node. Eq. (6.4) maps the differences between frequencies in the distribution to a percentage domain so that missing all nodes of a particular frequency is equally detrimental regardless of the number of those nodes in the target graph. However, due to its nature (6.4) is also the most difficult objective to satisfy given that two distributions must be identical to receive a perfect score with a linear fitness gradient, with respect to the number of different degree frequencies (in some cases this is less useful, e.g., a network entirely made up of nodes of degree 3 and one of 4 may function very similarly to a network of nodes of degree 3 and two of degree 4).

Although the raw fitness values are used during evolution and selection for all methods, for presentation they are mapped to the range $[0, 1]$ with a value of 1 being the most desirable. These adjusted fitnesses are computed as follows:

Maximize:

$$A_1 = \left[1 + \left(\frac{F_1 Raw}{n}\right)\right]^{-1} \tag{6.5}$$

$$A_2 = [1 + |C(G_t) - C(G_a)|]^{-1} \tag{6.6}$$

$$A_3 = 1 - \frac{F_3}{n} \tag{6.7}$$

$$A_4 = 1 - F_4 Raw \tag{6.8}$$

If a weighted sum fitness is used, objectives are summed to $F = \sum_{i \in O} w_i \times F_i Raw$, where O is the set of objectives and w_i is the weight of objective i, F is the fitness value assigned and used for selection in this case. If summed-ranks is used, then the raw fitnesses $F_i Raw$ are used as the values F_i to compute the ranks $Rank_{ji}$ described in (5.1). The rank weights are set to 1 for a normalized sum of ranks, or they can be set by the adaptive weighting mechanism (5.2). If Pareto ranking is used, as described in Chapter 5 an individual's Pareto rank, derived from the fitness values $F_i Raw$, is used for selection.

Before the fitness objective values were computed for any graph produced by an evolved model, the generated graph was first simplified to remove any self-loops or multi-edges. When calculating the average geodesic path length, if there was no path between a vertex pair the length of the path was returned as the size of the vertex set, a value larger than any possible path. If the graph was empty, the worst possible fitness values were assigned. If the user chose for the evolved model to be executed more than once during evaluation then the model was assigned the average fitness value per objective.

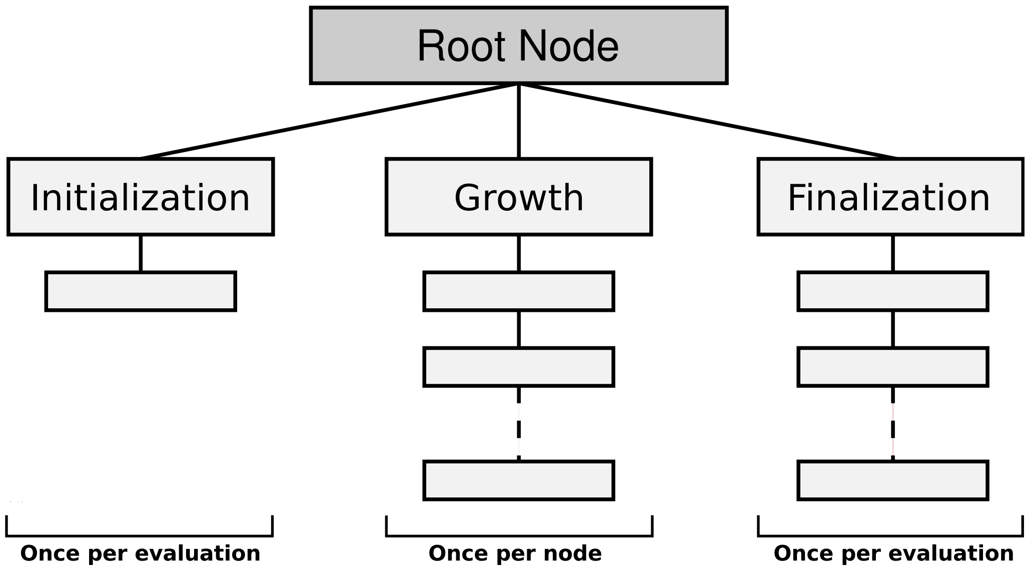


Figure 6.2: The shape of the trees used by the GP

6.3 GP Language

Strong typing [61] was used to enforce a specific tree shape, which was evaluated in the following manner. The root node has three branches, each containing a list of actions: one branch for initialization, one branch that defines growth actions, and one branch that describes finalization actions. Operations in the *initialization* branch are responsible for adding vertices or specifying how vertices are to be added during the graph building process. These actions are executed once per evaluation. *Growth* operations are responsible for adding edges to the graph, and are executed n times per evaluation, where n is the desired number of vertices in the generated network. *Finalization* operations are any operations that require edges and vertices to be present in the graph; for example, edge removal. Figure 6.2 illustrates the tree structure.

The three branches of the tree structure were based on a conceptualization that the construction of most anything could require some overhead for initialization or setup, followed by the construction process, and finally once the construction is complete may require some modifications. For example, the Watts-Strogatz model requires an initial network lattice to be constructed before the rewiring process begins.

While the tree model used by the GP is intuitive, there is at least one assumption implied by the GP tree structure, which is that the target network can be described as homogeneous in the process that constructed it. However, this is also an assumption made by each of the human designed models discussed here and it would be possible to extend the tree definition such that it represented a model of models.

The GP language used to construct the tree-based model includes a basic set of math operators $\{+, -, *, \%\}$, Ephemeral Random Constants (ERC), boolean functions, IF structures, and the relational $<$ operator that takes two float arguments and returns a boolean (only the $<$ operator was included because swapping the arguments is equivalent to flipping the operator). It also includes a number of other functions that were tailored to generate graphs, sometimes in a probabilistic way. The GP language functions are defined in the following list:

- $L = \{0.01, 0.02, 0.025, 0.05, 0.1, 0.2, ..., 1.0\}$.
- $P(index)$ takes an integer value as an argument and returns a floating point value from L at the location specified by $index\ MOD\ |L|$, where $|L|$ is the length of the list. If the value of $index$ is negative then the absolute value is used.
- $P(float)$ takes a floating point value and returns a value between $[0, 1]$. If the argument's absolute value is already within that range, it simply returns the absolute value of the argument. If the absolute value of the argument is greater than one then it computes $\lfloor |float| \rfloor$ and behaves in the same way as $P(index)$.
- ERC : An ephemeral random constant, initialized to a random floating point value which remains constant throughout the GP run.
- Math
  - $+$: Takes two floating point numbers and returns their sum.
  - $-$: Takes two floating point numbers and returns their difference.
  - $*$: Takes two floating point numbers and returns their product.
  - $\%$: Divides the left argument by the right argument and returns the result, if the denominator is zero, then one is returned.
- Boolean
  - $TRUE$: A statement of truth.
  - $FALSE$: Not TRUE.
  - AND : Returns TRUE if both operands resolve to TRUE.
  - OR : Returns TRUE if at least one operand resolves to TRUE.
- Relational
  - $<$: Takes two floating point numbers and returns TRUE if the left operand is less than the right one.
- Probability
  - `prob addProb`: Adds two probabilities, and clamps to the range $[0, 1]$ using the function $P(float)$.
  - `prob subProb`: Subtracts two probabilities, and clamps to the range $[0, 1]$ using the function $P(float)$.
  - `prob floatToProb`: Converts an arbitrary floating point value to a probability, using the function $P(float)$.

  - `prob indexToProb`: Converts an integer value to a probability, using the function $P(index)$.

- Structure Nodes
  - `INIT_LIST INIT_LIST(INIT_ACTION)`: Structure to capsulize an initialization action.
  - `GROW_LIST GROW_LIST(GROWTH_ACTION, GROW_LIST)`: Structure to chain together growth actions.
  - `FINISH_LIST FINISH_LIST(FINISH_ACTION, FINISH_LIST)`: Structure to chain together finalization actions.

- Initialization Actions
  - `INIT ADD_ALL_NODES`: Add all nodes to the graph with no edges.
  - `INIT BUILD_RING`: Add all nodes and build a ring.
  - `INIT SET_GROW_NODES`: One node is added at each iteration (grow).

- Growth Actions
  - `GROW CREATE_TRIANGLE`: Creates a triangle that includes the current node in the active graph.
  - `GROW CONNECT_W_PROB(prob)`: Connect to each node with probability *prob*.
  - `GROW CONNECT_RAND`: Connects the current node in the active graph to some random node in the active graph.
  - `GROW CONNECT_STUB(prob, bool)`: Connects an edge from the current node to a node that has previously executed `CONNECT_STUB`. The node that will be connected to is selected from the front *prob* portion of a priority queue arranged by node degree, either randomly if the second boolean argument is false, or probabilistically according to the degree of the node in the queue. Once a node has been connected to, it is removed from the queue of available nodes. If no nodes were available to the calling node then it is entered into the request queue of nodes available for connection.
  - `GROW CONNECT_STUB_PERSIST(Prob, bool)`: Connects an edge as `CONNECT_STUB` does, but it always adds its own request to the request queue and it does not remove any requests it satisfies from the request queue.
  - `DUPLICATE(prob)`: Connects the current node to the neighbours of a randomly selected node. With probability p the current node is also connected to the selected node. This is based on models of protein interaction and this behaviour is known to generate an exponential degree distribution [93].

- Branching and Conditional

  - `GROW_LIST if(bool, GROW_LIST, GROW_LIST)`: Executes the second argument if *bool* $= TRUE$, otherwise the third argument is executed.
  - `TRUE_WITH_PROB(prob)`: Returns TRUE with with probability *prob*.

- Finalization Actions
  - `FINAL REWIRE_EQUAL_PROB(prob)`: Rewire all edges with probability *prob*.
  - `FINAL REWIRE_RANDOM`: Randomly rewire edges.
  - `FINAL REMOVE_PROB(prob)`: Remove each edge with probability *prob*.

- Terminals
  - `prob baseIndexM`: An integer of type `index`, whose value is $M = \{1, 3, 5\}$.
  - `prob baseProb001`: An float of type `prob`, whose value is 0.001.
  - `float CUR_NODE_DEGREE`: Returns the current node degree as a float.
  - `float AVG_DEGREE`: Returns the average node degree in the active graph as a float.
  - `float MAX_DEGREE`: Returns the max node degree in the active graph as a float.
  - `float TOTAL_VERTEX_COUNT`: Returns the total vertex count in the active graph as a float.
  - `float FINAL_VERTEX_COUNT`: Returns the final expected vertex count in the active graph.
  - `float FINAL_EDGE_COUNT`: Returns the edge count in the active graph.
  - `GROWTH_ACTION NO_GROW`: A null grow action.
  - `GROWTH_ACTION NO_FINISH`: A null finalization action.
  - `GROW_LIST EMPTY_GROW_LIST`: Specify no growth actions.
  - `GROW_LIST EMPTY_FINISH_LIST`: Specify no finalization actions.

The motivation for each operator came from isolating important components of human-designed models and by a consideration of the features considered important in describing complex networks (the same ones used by the fitness functions). The `BUILD_RING`, `ADD_ALL_NODES`, `SET_GROW_NODES` functions exhibit graph initialization behaviour that has been shown to have a measurable impact on graph dynamics [7,97]. Probabilistic connections are utilized because they provide a method for tuning node degrees, `CREATE_TRIANGLE` was created because it can alter the clustering coefficient, and the `CONNECT_STUB` functions can produce preferential attachment behaviour, or produce disjoint sets of connected pairs. The finalization actions that rewire edges were inspired by the rewire process used in the Watts-Strogatz model [97] and the functions to remove edges can control network edge density.

Whatever the motivations may have been for adding each function, there are implications of their interactions that are not obvious. For example, if the `CREATE_TRIANGLE` function is applied repeatedly in conjunction with the `SET_GROW_NODES` operation it will create an exponential degree distribution – measured on average over 50 graphs of 500 nodes each, $\alpha \approx 2.98$, using the power law fitting method described in Chapter 3.

Chapter 7

Reproducing Existing Models

In order to examine the efficacy of the approach, graph models were evolved to match target networks generated by the ER, WS, and BA models described in Chapter 4. Generating models for known algorithms allows a direct comparison of the final evolved model to the known model, providing a strong validation. A graph was generated with 200 nodes by each model and these graphs were then used as input to the GP system. These graphs are called the *target graphs*. The GP system was run 50 times for each target graph, producing a set of candidate models.

A single evolved model must be chosen from the set of candidates for further experimentation and we would like to choose a model that performs well with respect to all fitness objectives. However, to avoid over-fitting it may be advantageous to select a model which performs somewhat worse in some objectives than other models, but much better than most of the candidates with respect to a few objectives. To select a model that fits this requirement, the following post-processing step was used for automatic model selection.

The final evolved models were used to generate 50 graphs that were in turn compared to the target graph. The comparisons were used to assign a final fitness value to each model with respect to each objective, i. The models were then ranked according to these values. The rankings were then automatically assigned a weight, α_i, based on the variance between average objective values across all candidate models:

$$\alpha_i = \frac{\max(\mu_{ji}) - \min(\mu_{ji})}{\sum_{i=1} \max(\mu_{ji}) - \min(\mu_{ji})} \tag{7.1}$$

where μ_{ji} is the average fitness objective produced by the j^{th} evolved model. Each evolved model j was then assigned a final weighted rank sum, $RankSum_j$, used to automatically select a final model for further experimentation (note that this value is used only for final model selection and not as a fitness value during evolution). $RankSum_j$ is computed as:

$$RankSum_j = \sum_i \alpha_i \times rank_{ji} \tag{7.2}$$

where $rank_{ji}$ is the rank of the j^{th} individual with respect to the i^{th} objective. The motivation for this method came from an information theory idea that the fewer times a value occurs the more information it provides and the more valuable it is. By selecting a model that generates features that the others are incapable of, it is more advantageous than selecting a model that does well in one or many objectives that all candidates are able to produce reasonably well. Once the ranking process

Table 7.1: Symbolic program simplification rules. The * matches any string.

Rule	Description
$(T \mid *)$ or $(* \mid T) = T$	If either argument to the boolean OR function is TRUE then replace the operator with TRUE.
$(F \,\&\, *)$ or $(* \,\&\, F) = F$	If either argument to the boolean AND function is FALSE then replace the operator with FALSE.
IF(F) BRANCH1, BRANCH2	If the boolean argument to an IF is FALSE then replace the operator with BRANCH2.
IF(T) BRANCH1, BRANCH2	If the boolean argument to an IF is TRUE then replace the operator with BRANCH1.
LIST(NO*, TAIL)	If the list head specifies no action, replace the operator with TAIL.

was completed the best model was selected for growth experiments.

Growth experiments measure how well the model performs at the task of projecting the growth of a complex network over time. This is done by comparing graphs produced by the evolved model at various sizes to graphs produced at the same sizes by the known target algorithm. The networks produced are compared with respect to the fitness measures, the *average Geodesic Path length Difference* (GPD), the difference in global clustering coefficients or the *Global Transitivity Difference* (GTD), the *KS-test statistic* (KS), and the *Normalized Sum of Percent Differences* (SPD).

The following sections describe how a model was evolved for the random graph model, the Watts-Strogatz model, and the Barabási-Albert model. All experiments were conducted using the parameters specified in Table 7.2, which were established empirically and found to produce good results for all target model types. The results are presented in the form of degree distribution plots for the growth experiments, and pseudocode of the evolved algorithms where it is helpful, as well as tables containing the numeric comparison of fitness values. In the cases where pseudocode is given for the evolved models, the models were first simplified symbolically by the system using a set of simple rules which are defined in Table 7.1. Once this process was complete the output was then further simplified by hand and translated from the GP language into more readable pseudocode.

7.1 Handling Multi-Objectivity

Given that multiple objectives were used to judge the fitness of graph models during evolution, and that different methods for handling the objectives may effect the optimization process, a comparison of different methods was performed to establish which method yields the best results for the input graphs generated by the Erdös-Rényi, Watts-Strogatz, and Barabási-Albert models. The comparison includes a weighted sum with equal objective weights (WSum) because of its simplicity; a weighted sum

Table 7.2: GP parameters.

Parameter	**Value**
Initialization Method	Grow
Grow Min	3
Grow Max	5
Maximum Tree Size	17
Population Size	100
Generations	50
Selection	Tournament: k=3
Crossover	Subtree Crossover: 0.95
Mutation	Grow: 0.2, linearly decreasing, $\text{min depth} = 1, \text{max depth} = 4$
Runs	50
Executions per eval.	10

with adaptive weighting (AWSum) because equal weights may not be suitable and a manual search of weights is time consuming; normalized sum of ranks (NSR) as a simple multi-objective approach that does not require weighting; normalized sum of ranks with adaptive weighting as a biased multi-objective approach meant to promote less common solutions; finally, Pareto ranking because it is a common multi-objective approach [36]. The adaptive weighting used for AWSum and ANSR is described by (5.2).

The comparison was performed by evolving a set of graph models for each input graph by each of the different methods for handling multiple fitness objectives. When a GP run has been completed using WSum or AWSum, a single model is output, NSR and ANSR will output all of those that share the highest rank sum (usually one), and Pareto will output the entire non-dominated set in the population. Fifty GP runs using the parameters given in Table 7.2 were conducted per method per input target graph for 200 node graphs.

The best models produced by each method were compared per input graph type with respect to their fitness as computed over 50 evaluations against the target. To choose the 'best' model with respect to the target type and the optimization method, the fitness of each model was given a rank that was normalized by dividing by the maximum rank in that objective. The rank sum was then computed and models were sorted by the rank sum where a rank one model is considered the best. One model is selected at the end of this process for each network target and multi-objective fitness evaluation strategy.

The fitness values across optimization methods were computed within each input target group via a non-parametric one-way analysis of variance [51]. The test showed there were significant differences with 95% confidence for all methods for all input classes except for the case of the BA input types (making 8 significant comparisons). Next, a non-parametric version of Tukey's HST test [51, 59] was used post-hoc to discern which optimization methods had significantly different means outside of a 95%

	GTD	**GPD**	**KS**	**SPD**
ER	$1.23 \cdot 10^{-30}$	$1.15 \cdot 10^{-20}$	$9.12 \cdot 10^{-17}$	$2.51 \cdot 10^{-2}$
BA	-	-	-	-
WS	$2.08 \cdot 10^{-116}$	$2.39 \cdot 10^{-45}$	$2.03 \cdot 10^{-16}$	$1.63 \cdot 10^{-8}$

Table 7.3: The p-values for the non-parametric one-way test. There was not enough variance to report p-values for the BA results. The acronyms GTD, GPD, KS, and SPD correspond to the global transitivity difference, geodesic path difference, KS test statistic (D-value), and the sum of percent differences.

confidence interval within those comparisons deemed significant by the one-way test. Figures 7.1, 7.2, and 7.3 show box plots of the performance data per multi-objective strategy, per fitness objective, per input type – the mean value is represented as the dark horizontal line, while the interquartile range forms the upper and lower edges of the box, the vertical lines indicate the maximum and minimum values while the dots indicate outliers. Table 7.3 shows the p-values from the one-way test, while Table 7.4, to 7.10 show the results of the post-hoc test. The post-hoc tables are read such that the cell at row i and column j indicates the number of times method j was significantly better than method i (out of 50 comparisons). The post-hoc test was not able to identify which differences were significant within the ER data for the SPD objective (corresponding to the largest p-value in 7.3), so the table was omitted.

Table 7.11 is a summary of the previous post-hoc tables where the cell at row i and column j indicates the fraction of times method j was significantly better (or negative if it was worse) than method i across all objectives. The final row gives the column sum, and indicates the quality of the column method versus all other methods (a negative value means it was generally worse, while a positive value means it was generally better), the larger the number the better the method. Finally, Table 7.12 was constructed in the same way as Table 7.11, only the values within the table were normalized by the number of significant comparisons. The last row in Table 7.12 continues to give the column sum.

Based on the comparison of multi-objective fitness strategies WSum consistently produced the best models given the set of GP parameters, target graphs, and selection criteria for comparison used. Although weight selection for WSum can be problematic, equal weighting in this context does well, and the WSum approach is very simple. The remainder of the results presented here will thus use the WSum method.

7.2 Model Inference From A Random Graph

A model was evolved over 50 generations, with a population size of 100 individuals, against a 200 node target graph generated using the Erdös-Rényi model, with $p = 0.05$. All other parameters are the same as those given in Table 7.2. The best model, ER_{200} (that was selected using the ranking system previously described). The evolved model made use of the `ADD_ALL_NODES` initialization function and it made heavy use of the `CONNECT_W_PROB` function. Following this, it then removed

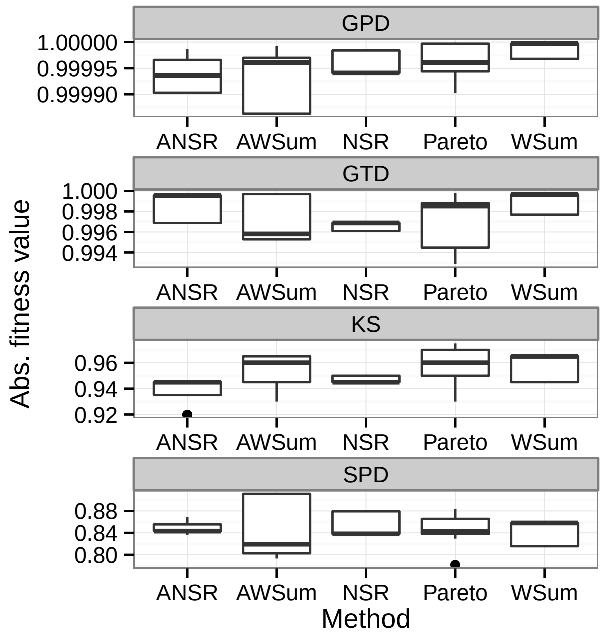


Figure 7.1: Box plots of ER input data, 50 graphs from the best result produced by each multi-objective technique.

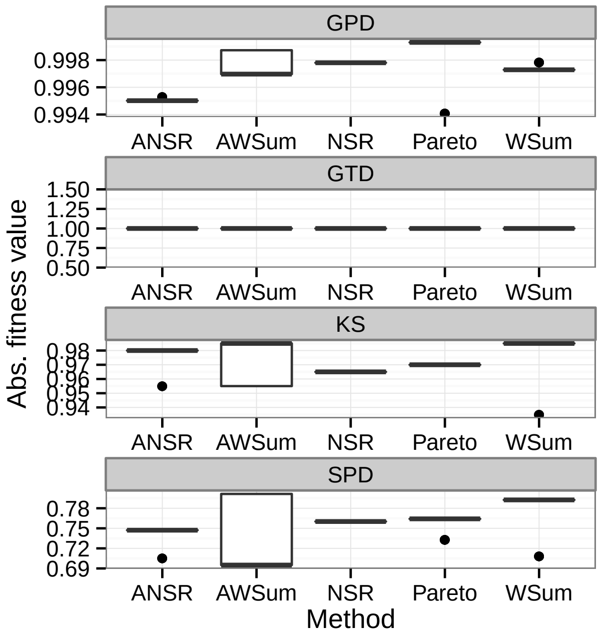


Figure 7.2: Box plots of BA input data, 50 graphs from the best result produced by each multi-objective technique.

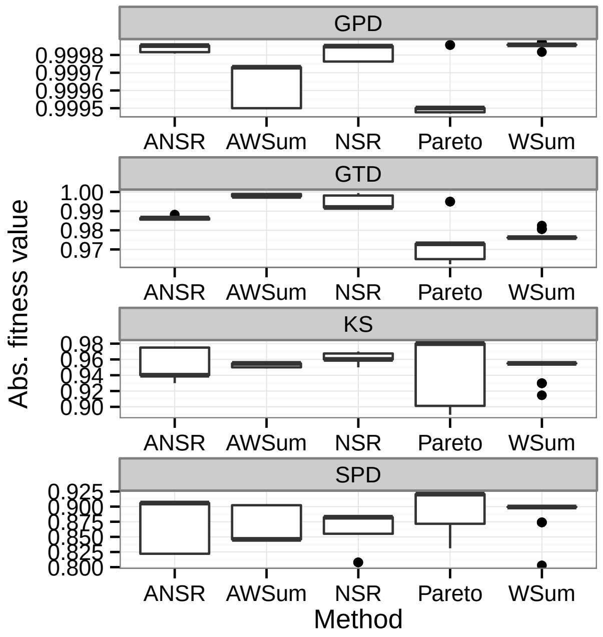


Figure 7.3: Box plots of WS input data, 50 graphs from the best result produced by each multi-objective technique.

ER input data, GTD objective					
	ANSR	AWSum	NSR	Pareto	WSum
ANSR	0	0	-1	-1	1
AWSum	0	0	0	0	0
NSR	1	0	0	0	1
Pareto	1	0	0	0	1
WSum	-1	0	-1	-1	0

Table 7.4: Comparison of multi-objective techniques on ER data for the GTD objective. The value at row i and column j indicates how many times method j was better than method i.

ER input data, GPD objective					
	ANSR	AWSum	NSR	Pareto	WSum
ANSR	0	0	1	1	1
AWSum	0	0	0	0	1
NSR	-1	0	0	1	1
Pareto	-1	0	-1	0	1
WSum	-1	-1	-1	-1	0

Table 7.5: Comparison of multi-objective techniques on ER data for the GPD objective. The value at row i and column j indicates how many times method j was better than method i.

ER input data, KS objective					
	ANSR	AWSum	NSR	Pareto	WSum
ANSR	0	1	1	1	1
AWSum	-1	0	-1	0	0
NSR	-1	1	0	1	1
Pareto	-1	0	-1	0	0
WSum	-1	0	-1	0	0

Table 7.6: Comparison of multi-objective techniques on ER data for the KS objective. The value at row i and column j indicates how many times method j was better than method i.

WS input data, GTD objective					
	ANSR	AWSum	NSR	Pareto	WSum
ANSR	0	1	1	-1	-1
AWSum	-1	0	-1	-1	-1
NSR	-1	1	0	-1	-1
Pareto	1	1	1	0	1
WSum	1	1	1	-1	0

Table 7.7: Comparison of multi-objective techniques on WS data for the GTD objective. The value at row *i* and column *j* indicates how many times method *j* was better than method *i*.

WS input data, GPD objective					
	ANSR	AWSum	NSR	Pareto	WSum
ANSR	0	-1	-1	-1	1
AWSum	1	0	1	0	1
NSR	1	-1	0	-1	1
Pareto	1	0	1	0	1
WSum	-1	-1	-1	-1	0

Table 7.8: Comparison of multi-objective techniques on WS data for the GPD objective. The value at row *i* and column *j* indicates how many times method *j* was better than method *i*.

WS input data, KS objective					
	ANSR	AWSum	NSR	Pareto	WSum
ANSR	0	1	1	0	0
AWSum	-1	0	1	0	0
NSR	-1	-1	0	0	-1
Pareto	0	0	0	0	0
WSum	0	0	1	0	0

Table 7.9: Comparison of multi-objective techniques on WS data for the KS objective. The value at row *i* and column *j* indicates how many times method *j* was better than method *i*.

WS input data, SPD objective					
	ANSR	AWSum	NSR	Pareto	WSum
ANSR	0	0	0	1	0
AWSum	0	0	0	1	0
NSR	0	0	0	1	1
Pareto	-1	-1	-1	0	0
WSum	0	0	-1	0	0

Table 7.10: Comparison of multi-objective techniques on WS data for the SPD objective. The value at row i and column j indicates how many times method j was better than method i.

	ANSR	AWSum	NSR	Pareto	WSum
ANSR	0	2	2	0	3
AWSum	-2	0	0	0	1
NSR	-2	0	0	1	3
Pareto	0	0	-1	0	4
WSum	-3	-1	-3	-4	0
Col. Sums:	**-7**	**1**	**-2**	**-3**	**11**

Table 7.11: A summary of the post-hoc tables. The larger the number in the last row, the better the performance of the column method.

	ANSR	AWSum	NSR	Pareto	WSum
ANSR	0.000	0.250	0.250	0.000	0.375
AWSum	-0.250	0.000	0.000	0.000	0.125
NSR	-0.250	0.000	0.000	0.125	0.375
Pareto	0.000	0.000	-0.125	0.000	0.500
WSum	-0.375	-0.125	-0.375	-0.500	0.000
Col. Sums:	**-0.875**	**0.125**	**-0.250**	**-0.375**	**1.375**

Table 7.12: A summary of the normalized post-hoc tables. The larger the number in the last row, the better the performance of the column method.

Algorithm 2 ER_{200} algorithm

```
ADD_ALL_NODES; {Add all nodes with no edges.}
for each node $n$ do
    CONNECT_W_PROB(0.05);
    CONNECT_STUB(prob_0.01), FALSE);
end for
REMOVE_PROB(0.05);
```

edges it had added using the `REMOVE_PROB` function using a probability of 0.09. Pseudocode for the ER_{200} algorithm is given in Alg. 2. The evolved algorithm achieves the random graph model behaviour by initializing the graph with all nodes using `ADD_ALL_NODES` utilizing the `CONNECT_W_PROB` function, adding additional edges with `CONNECT_STUB` and finally uses `REMOVE_PROB` to remove a small portion of edges. Fig. 7.4 to Fig. 7.7 are degree distribution comparison plots for graphs at 200, 400, 600, and 800 nodes respectively. The model ER_{200}, evolved against a 200 node target, was used to generate 50 graphs at each size, which was then compared against a graph generated at the same size by the Erdös-Rényi algorithm. The distributions labelled as "Evolved Model" are the average degree distributions of the 50 graphs generated by ER_{200}. The average distributions have error bars at one standard deviation about the means. These plots show that the average degree in the graphs produced by both ER_{200} and the Erdös-Rényi model both increase with the number of nodes in the graph and both produce graphs containing nodes with similar degrees with similar frequencies. The mean p-values for KS tests comparing distributions for each graph produced by the evolved model against the corresponding target graph are given below the distribution plots. The results of the KS tests show that the distributions are the same with a mean confidence of between 0.7851 and 0.828 depending on the size of the graph, although the best matches have corresponding p-values not worse than 0.9425 regardless of graph size.

The results of the comparison with respect to the other fitness measures are given in Table 7.13. A value of 1.0 is best and a value of 0.0 is worst. All average values are at least 0.9 regardless of the size of the graph produced, except the value of the SPD, which is the most difficult fitness objective (consider that in order to achieve a SPD value of 1.0 the degree distributions of all 50 graphs generated by the evolved model would need to match the target distribution perfectly). The small amount of variation in the fitness values, regardless of the output network size, indicates that the evolved model continues to perform well even when generating graphs at four times the size of the network used as input during evolution.

The evolved model and the target model also generate similar edge densities. Given that the evolved model also uses only probabilistic connections this means The probability of edge occurrences should also be similar. The edge probability can be computed as $p = m\binom{n}{2}^{-1}$ in a random graph [74], and this value was computed for all graphs of all sizes generated by ER_{200}. The value was found to be $0.047 \leq p \leq 0.05$, which includes the probability of 0.05 used to construct the target model.

Table 7.13: Comparison of graphs produced by ER_{200} to graphs produced by the Erdös-Rényi model. Size refers to the number of nodes, μ contains average values, σ contains standard deviations.

Size	Measure	Min	μ	Max	σ
200	GPD	1.000	1.000	1.000	$3.86 \cdot 10^{-5}$
	GTD	0.997	0.998	1.000	$8.67 \cdot 10^{-4}$
	KS	0.935	0.939	0.950	$5.44 \cdot 10^{-3}$
	SPD	0.776	0.793	0.820	$2.11 \cdot 10^{-2}$
400	GPD	1.000	1.000	1.000	$6.61 \cdot 10^{-6}$
	GTD	0.995	0.997	0.998	$9.67 \cdot 10^{-4}$
	KS	0.925	0.942	0.958	$1.27 \cdot 10^{-2}$
	SPD	0.756	0.774	0.791	$1.30 \cdot 10^{-2}$
600	GPD	1.000	1.000	1.000	$2.74 \cdot 10^{-6}$
	GTD	0.995	0.997	0.998	$5.33 \cdot 10^{-4}$
	KS	0.938	0.950	0.962	$6.78 \cdot 10^{-3}$
	SPD	0.725	0.754	0.779	$1.38 \cdot 10^{-2}$
800	GPD	1.000	1.000	1.000	$1.43 \cdot 10^{-6}$
	GTD	0.996	0.997	0.997	$3.73 \cdot 10^{-4}$
	KS	0.949	0.963	0.974	$5.79 \cdot 10^{-3}$
	SPD	0.728	0.751	0.788	$1.47 \cdot 10^{-2}$

Averaged over 50 graphs per size. Similarity measures are defined in (6.5), (6.6), (6.7), and (6.8).

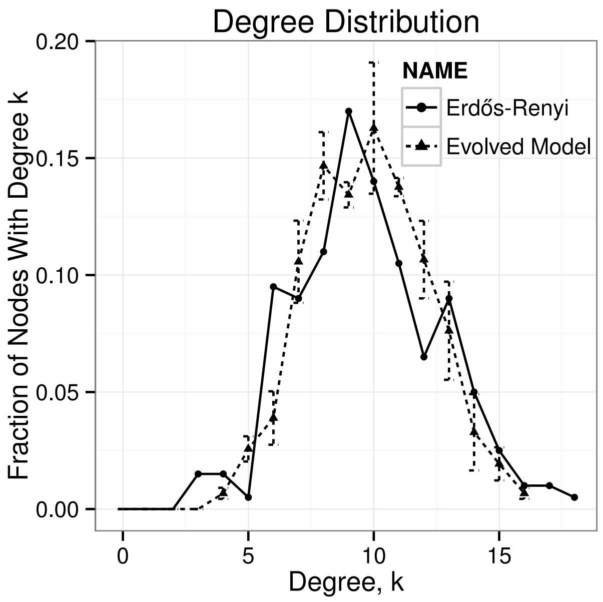


Figure 7.4: ER_{200} vs. Erdös-Rényi degree distribution, $n = 200$. Mean KS test statistic, $D = 0.204$. Mean p-value $p = 0.779$.

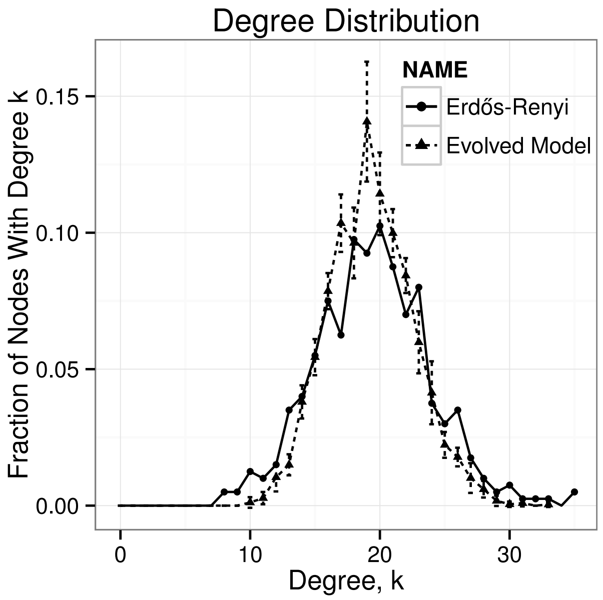


Figure 7.5: ER_{200} vs. Erdös-Rényi degree distribution, $n = 400$. Mean KS test statistic, $D = 0.152$. Mean p-value $p = 0.806$.

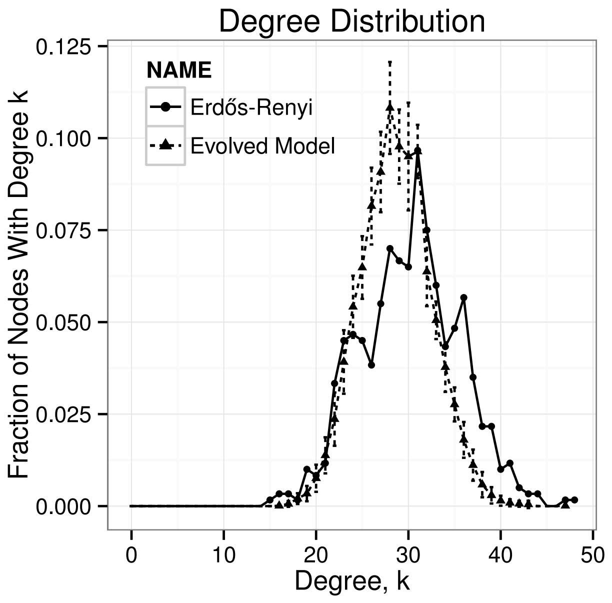


Figure 7.6: ER_{200} vs. Erdös-Rényi degree distribution, $n = 600$. Mean KS test statistic, $D = 0.127$. Mean p-value $p = 0.836$.

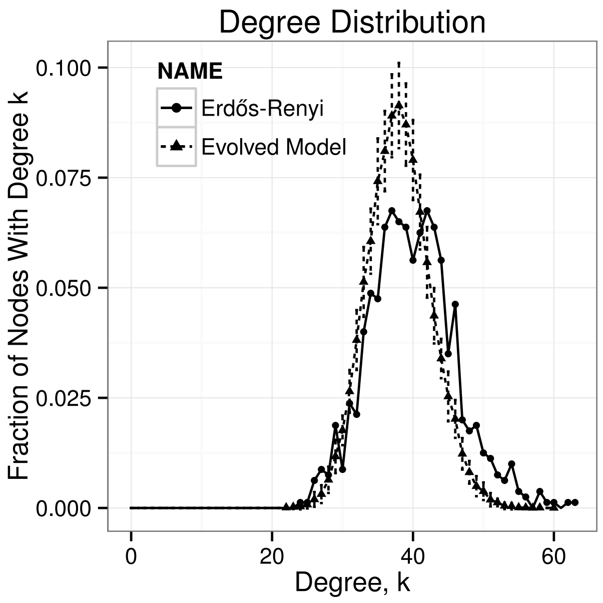


Figure 7.7: ER_{200} vs. Erdös-Rényi degree distribution, $n = 800$. Mean KS test statistic, $D = 0.117$. Mean p-value $p = 0.807$.

7.3 Model Inference From A Watts-Strogatz Graph

Next, a model was evolved against a 200 node Watts-Strogatz target graph over 50 generations, using a population size of 100 individuals. All other parameters are the same as those given in Table 7.2. The target graph was generated with a rewiring probability $p_{rew} = 0.15$.

Pseudocode for the SW_{200} algorithm is not reproduced here because the evolved model contained 171 different functions and terminals and 18 conditional branches. However, it can be found in Appendix B. Looking at only the 19 functions that add or remove edges and nodes to the graph some observations can be made. The SW_{200} model uses the `BUILD_RING` function to initialize a graph, this behaviour is consistent with the Watts-Strogatz model, it also contains 6 instances of `CREATE_TRIANGLE`, 4 instances of `CONNECT_RAND`, and 8 instances of `CONNECT_STUB`. The construction of triangles influences the clustering coefficient and edges created by `CONNECT_RAND`, and `CONNECT_STUB` provide shortcuts across the ring. The finalization branch was empty. The evolved algorithm will produce graphs with a mean degree of approximately 6 ± 1 for any size graph. A comparison of the clustering coefficient values of graphs produced by the evolved model versus the clustering coefficient for a random graph with the same number of nodes and edges is given in Table 7.14. Note that even the minimum values for the evolved model are an order of magnitude higher than the random graph values, so the values observed in the evolved model cannot be attributed to random chance.

Table 7.14: Clustering coefficient comparison. The first and third quartiles are notated as Q1 and Q3 respectively.

Size	**Min**	**Q1**	**Median**	**Q3**	**Max**	**Random**
200	0.181	0.193	0.203	0.203	0.203	$1.49 \cdot 10^{-2}$
400	0.172	0.172	0.189	0.185	0.193	$2.01 \cdot 10^{-2}$
600	0.161	0.164	0.174	0.174	0.185	$3.02 \cdot 10^{-2}$
800	0.153	0.167	0.169	0.171	0.177	$6.02 \cdot 10^{-2}$

Fig. 7.8 to Fig. 7.11 are degree distribution comparison plots for graphs at 200, 400, 600, and 800 nodes respectively. The distributions labelled as "Evolved Model" are the average degree distributions of the 50 graphs generated by SW_{200}. The plots show that the evolved model and the target model have produced graphs with the same minimum k values, and a similar frequency of nodes with those values. Both the evolved and target models create degree distributions with maximum frequencies between 0.3 and 0.4 at $k = 5$ or $k = 6$ regardless of the number of nodes in the graph. This behaviour is very different from the random graph model and the ER_{200} model in the previous section that produced graphs with an average degree that increased with the number of nodes in the graph. The mean p-values for KS tests comparing distributions for each graph produced by the evolved model against the corresponding target graph are once again given below the distribution plots. The results of the KS tests show the distributions are the same with an average confidence of at least 97% in all instances. The results of the comparison with respect to the other fitness measures are given in Table 7.15. A value of 1.0 for each objective is best and a value of 0.0 is worst. The average values are at least 0.8. These values show that the quality of fit is maintained even when generating graphs much larger than the graph used as input to the GP system with respect to all objectives.

7.4 Model Inference From A Barabási-Albert Graph

Finally, a population of 100 was evolved over 50 generations against a target generated by the Barabási-Albert algorithm.

Pseudocode for the BA_{200} algorithm is given in Alg. 3. The evolved algorithm makes use of the `SET_GROW_NODES` initialization function that helps generate the branching structure. It achieves the preferential attachment behaviour by using the `CONNECT_STUB_PERSIST` function using a small range specification (only the top 2% of nodes in the priority queue will be selected), and a degree-proportionate selection from that range.

Fig.7.12 to Fig. 7.15 are degree distribution comparison plots for graphs at 200, 400, 600, and 800 nodes respectively. The distributions labelled as "Evolved Model" are the average degree distributions of the 50 graphs generated by BA_{200}. When the evolved model degree distributions are compared to the target graphs, it is observed

Table 7.15: Comparison of graphs produced by SW_{200} to graphs produced by the Watts-Strogatz model. Size refers to the number of nodes, μ contains average values, σ contains standard deviations.

Size	**Measure**	**Min**	μ	**Max**	σ
200	GPD	1.000	1.000	1.000	$2.09{\cdot}10^{-5}$
	GTD	0.970	0.987	0.991	$5.22{\cdot}10^{-3}$
	KS	0.920	0.927	0.940	$5.05{\cdot}10^{-3}$
	SPD	0.828	0.850	0.920	$3.11{\cdot}10^{-2}$
400	GPD	1.000	1.000	1.000	$6.68{\cdot}10^{-5}$
	GTD	0.963	0.975	0.982	$7.69{\cdot}10^{-3}$
	KS	0.825	0.844	0.873	$1.50{\cdot}10^{-2}$
	SPD	0.808	0.879	0.913	$4.51{\cdot}10^{-2}$
600	GPD	1.000	1.000	1.000	$2.71{\cdot}10^{-5}$
	GTD	0.948	0.960	0.972	$9.02{\cdot}10^{-3}$
	KS	0.835	0.868	0.885	$1.57{\cdot}10^{-2}$
	SPD	0.851	0.878	0.938	$3.04{\cdot}10^{-2}$
800	GPD	1.000	1.000	1.000	$2.14{\cdot}10^{-5}$
	GTD	0.942	0.958	0.972	$7.75{\cdot}10^{-3}$
	KS	0.861	0.874	0.901	$1.17{\cdot}10^{-2}$
	SPD	0.833	0.876	0.903	$2.40{\cdot}10^{-2}$

Averaged over 50 graphs per size. Similarity measures are defined in (6.5), (6.6), (6.7), and (6.8).

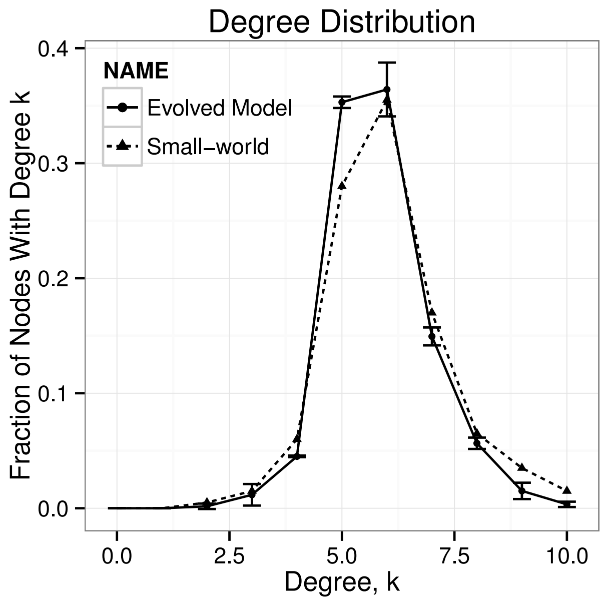


Figure 7.8: SW_{200} vs. Watts-Strogatz distribution, $n = 200$. Mean KS test statistic, $D = 0.105$. Mean p-value $p = 1.000$.

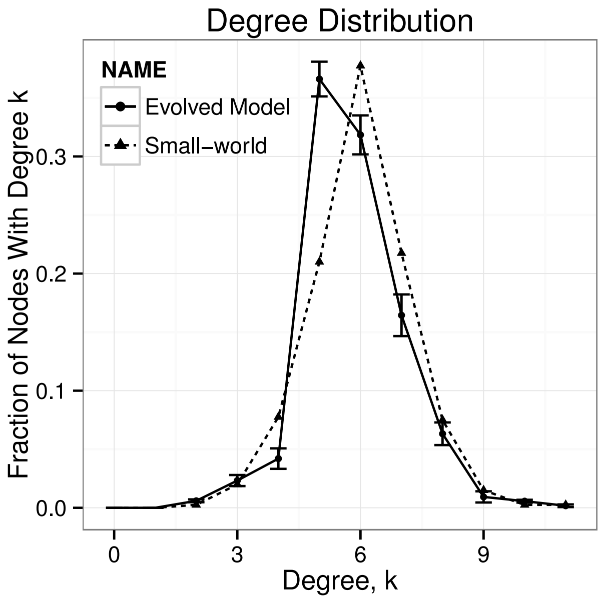


Figure 7.9: SW_{200} vs. Watts-Strogatz distribution, $n = 400$. Mean KS test statistic, $D = 0.105$. Mean p-value $p = 1.000$.

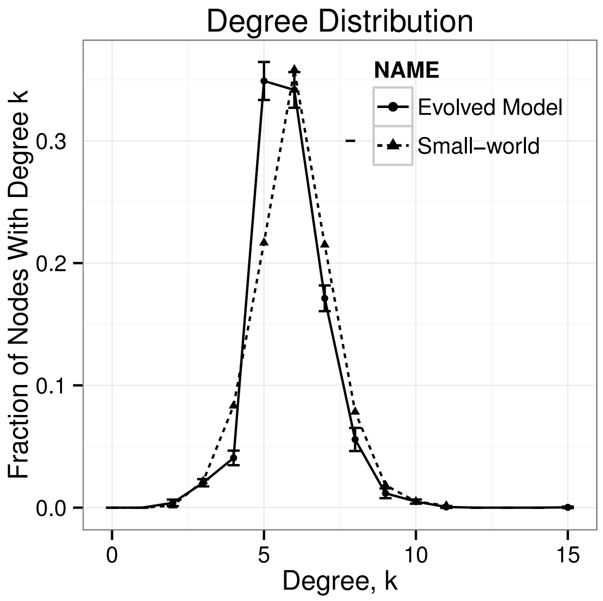


Figure 7.10: SW_{200} vs. Watts-Strogatz distribution, $n = 600$. Mean KS test statistic, $D = 0.132$. Mean p-value $p = 0.978$.

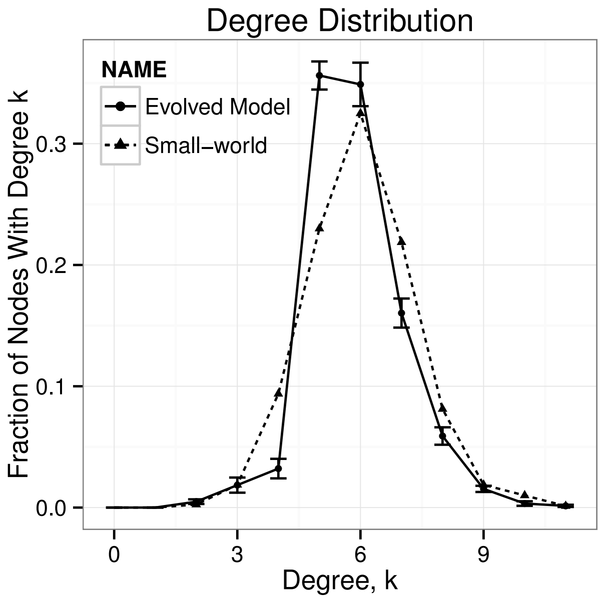


Figure 7.11: SW_{200} vs. Watts-Strogatz distribution, $n = 800$. Mean KS test statistic, $D = 0.091$. Mean p-value $p = 1.000$.

Algorithm 3 BA_{200} algorithm

```
SET_GROW_NODES; {Add one node per iteration.}
for each node $n$ do
    CONNECT_STUB_PERSIST(FLOAT_TO_PROB(0.02), TRUE);
end for
```

Table 7.16: Comparison of graphs produced by BA_{200} to graphs produced by the Barabási-Albert model. Size refers to the number of nodes, μ contains average values, σ contains standard deviations.

Size	**Measure**	**Min**	μ	**Max**	σ
200	GPD	0.997	0.999	1.000	$1.12{\cdot}10^{-3}$
	GTD	1.000	1.000	1.000	0
	KS	0.955	0.956	0.960	$2.27{\cdot}10^{-3}$
	SPD	0.725	0.734	0.737	$5.45{\cdot}10^{-3}$
400	GPD	0.996	0.996	0.997	$2.26{\cdot}10^{-4}$
	GTD	1.000	1.000	1.000	0
	KS	0.965	0.976	0.988	$1.14{\cdot}10^{-2}$
	SPD	0.659	0.706	0.753	$4.74{\cdot}10^{-2}$
600	GPD	0.999	1.000	1.000	$3.22{\cdot}10^{-4}$
	GTD	1.000	1.000	1.000	0
	KS	0.982	0.982	0.983	$8.24{\cdot}10^{-4}$
	SPD	0.717	0.749	0.786	$2.62{\cdot}10^{-2}$
800	GPD	1.000	1.000	1.000	$1.51{\cdot}10^{-4}$
	GTD	1.000	1.000	1.000	0
	KS	0.988	0.988	0.991	$1.62{\cdot}10^{-3}$
	SPD	0.709	0.725	0.736	$1.02{\cdot}10^{-2}$

Averaged over 50 graphs per size. Similarity measures are defined in (6.5), (6.6), (6.7), and (6.8).

that they have almost identical frequencies of low degree nodes and are both heavily tailed distributions with a small number of nodes with high degrees. The mean p-values for the KS test are given with these figures and show the distributions are the same with at least 95% confidence for all input graph sizes. The results of the comparison with respect to the other fitness measures are given in Table 7.16. A value of 1.0 for each objective is best and a value of 0.0 is worst. The comparisons show that the model performance is maintained even when generating graphs at 800 nodes. The values of 1.0 for the difference in global transitivity fitness objective reflects the strong tree-like structure of the graphs generated by the Barabási-Albert graphs and the corresponding BA_{200} model, neither model is very likely to produce any triangles at all.

7.5 Effective Difference Between Graph Properties

A models evolved by the GP system represents an estimate of the *true* model T that generated the target graph. The estimate is created based on a sample of nodes in a single graph target. We have also shown via the fitness functions that the evolved models are able to provide a good approximation of much larger graphs generated by T. However, we expect some differences between the evolved model

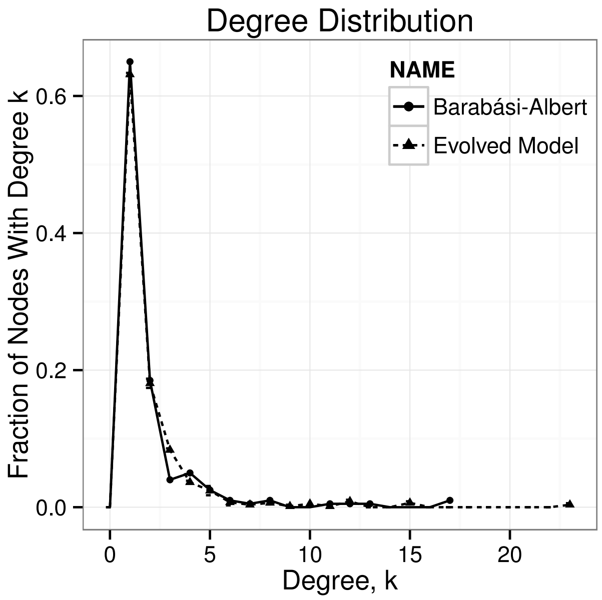


Figure 7.12: BA-M_{200} vs. Barabási-Albert distribution, $n = 200$. Mean KS test statistic, $D = 0.0825$. Mean p-value $p = 0.969$.

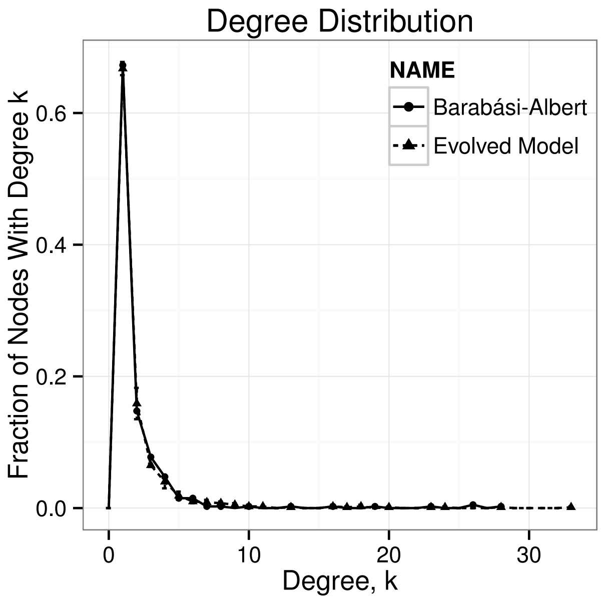


Figure 7.13: BA-M_{200} vs. Barabási-Albert distribution, $n = 400$. Mean KS test statistic, $D = 0.1136$. Mean p-value $p = 0.961$.

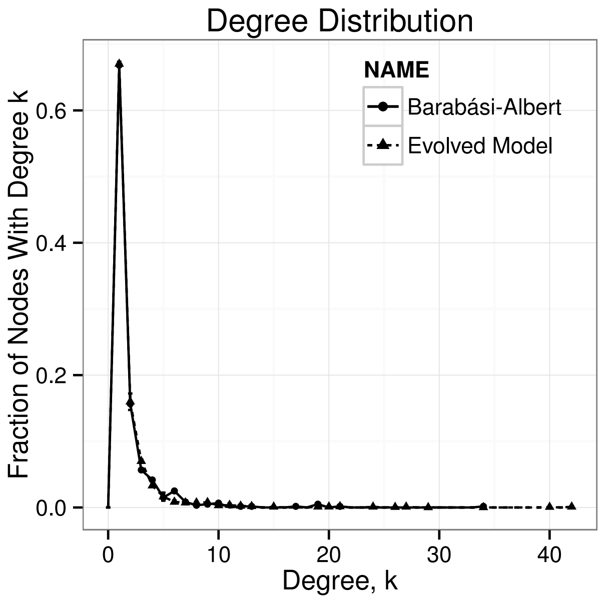


Figure 7.14: BA-M_{200} vs. Barabási-Albert distribution, $n = 600$. Mean KS test statistic, $D = 0.0563$. Mean p-value $p = 1.000$.

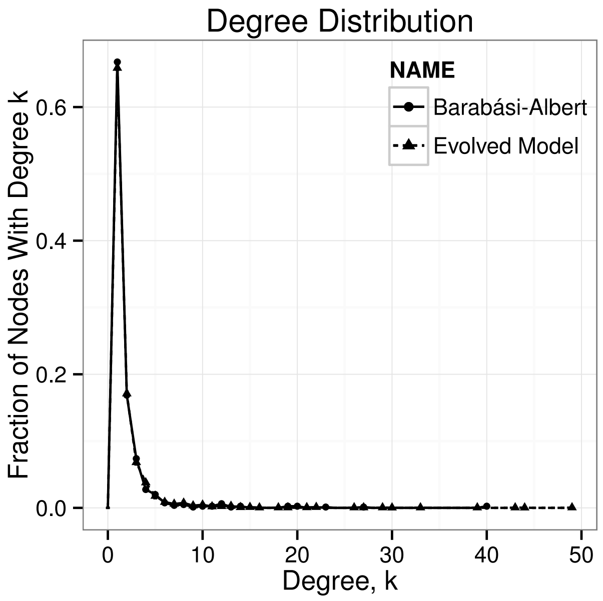


Figure 7.15: BA-M_{200} vs. Barabási-Albert distribution, $n = 800$. Mean KS test statistic, $D = 0.0712$. Mean p-value $p = 0.999$.

Table 7.17: Effect sizes of graph properties vs. targets, 1000 samples, Global clustering coefficient (GC), Avg. geodesic path length (GP).

Size	**Property**	g_{ER}	g_{WS}	g_{BA}
200	GP	0.009	0.044	0.060
	GC	0.005	0.035	0.000
400	GP	0.024	0.076	0.125
	GC	0.004	0.043	0.000
600	GP	0.028	0.089	0.142
	GC	0.004	0.044	0.000
800	GP	0.027	0.099	0.049
	GC	0.004	0.047	0.000

and T because the fitness functions show us the approximations are not optimal. Thus, the distribution of network characteristics is likely to differ, but as long as the effective difference is negligible then our model is very reasonable. We show that using Hedges g-statistic [40] the effect size between the evolved models and and T is small (much less than 0.2) [16], thus the models represent good approximations of T. Moreover, these small differences are practically negligible in the context of the graphs themselves. Table 7.17 gives the effect size of the distributions of average geodesic path length (GP) and global clustering coefficient (GC) between the evolved models ER_{200}, WS_{200}, and BA_{200} and the Erdös-Rényi, Watts-Strogatz, and Barabási-Albert models, respectively.

No g-statistics are given comparing degree distributions because they would first need to be collapsed to a single descriptive statistic. Given the various shapes of the degree distributions it makes no sense to compare them in that context. However, comparing the degree distribution is implicit in the KS test statistic, which is already known to have evolved to a good value based on the p-values given in the Sections 7.2 to 7.4.

7.6 In Summary

The experiments in the previous section show the proposed GP system is able to approximate known graph models given only a sample of the networks they generate as input. The results are validated by a manual comparison of algorithms (because in this case we know the target algorithm) as well as by a comparison of graph properties coupled with goodness of fit tests for their degree distributions and effect sizes for the remaining properties. Five different multi-objective strategies were compared and it was shown that a simple weighted sum of the objective functions was suitable for this problem, although different input networks may give different results.

Out of curiosity (it was never intended to be within the scope of this thesis), the system was applied to a cortical network which had no obviously suitable model [101]. The initial results were very promising, see Appendix C. However, the GP system could not generate models able to reproduce the defined community structure of the cortex, although neither could well-known models traditionally used to model cortical networks. With some alterations to the system, it was coerced into generating models of community structure and the following chapter explains how.

Chapter 8

Application to Cortical Networks

The experiments and results detailed here outline how a model for the cat cortex was generated. The model is significant on two levels. First, it is significant because it was automatically inferred by the proposed GP system, which was the first (to this author's knowledge) system for automatically inferring graph models for complex networks – making this network the first real network to have a graph model automatically inferred for it. Second, it is significant because the graph model which was inferred possesses important features which graph models traditionally used to approximate the cortex do not possess.

8.1 Cortical Networks

Cortical networks, having a low average geodesic path length and a high clustering coefficient are often modelled with the Watts-Strogatz (WS) small-world model [78, 89, 90, 97]. However, the WS model does not produce highly connected hub nodes, an important feature of brain structure. The hub nodes present in cortical networks have given rise to the idea that the connectivity may follow a power law. If it is necessary to model hub nodes then a model such as the Barabási-Albert (BA) is usually used [7, 78, 90]. Both of these models have been important in the analysis of brain networks [90], however neither of them model both the degree distribution as well as the transitivity and path length properties.

The experiments in this section will focus on the cortical connectivity within the brain of a cat because it has the most complete available data of its kind [101]. The dataset was created via a collation of tract-tracing experiments and literature reporting, supported by analytical treatment of the results [85]. It has been found that this network is dominated by paths of length one and two, that it contains a high density of connections, it has highly connected hubs, and that it is segregated into communities (which closely align with the visual, auditory, somatosensory-motor, and frontolimbic centres) [101]. It is also speculated that the number of alternative pathways between cortical nodes plays an important role in the brain's ability to process complex information [101]. Cortical networks are often classified as small-world networks [89, 101]; however, there is also some debate as to whether or not they could be scale-free networks [38, 45, 91]. As such, these networks are often modelled or simulated using algorithms that generate scale-free or small-world networks. Although, it is known that the cortical data in the literature suggests neither model is quite suitable given that they do not describe the inter-community dynamics of the cortex well [101].

The proposed system for the automatic inference of graph models works only with undirected networks, and while cortical networks are thought to be directed networks, it has been shown that an undirected approximation is a reasonable relaxation because the majority of connections are reciprocal [66, 101]. The cat cortical network dataset [85] in its undirected form contains 52 cortical nodes and 515 edges. It has a clustering coefficient of $C_c = 0.585$ and an average geodesic path length of $l_c = 1.636$ and a diameter of $d_c = 3$. These simple properties are useful because they are known to be good classifiers of the behaviour of message propagation across a network [6]. If multiple paths are used for processing then it may be that the longest path is also of consequence, especially if the temporal aspect of these signals is important, so the diameter will also be considered.

8.2 Generating Hierarchical Models of Community Structure

This thesis does not focus on the problem of discovering communities directly with the GP system itself, although it may be an effort for future work. Instead, we endeavour to model communities that we are able to identify in a target network via an external algorithm (many exist [25]). We propose a system which represents the first effort to automatically infer graph models for networks which exhibit a hierarchical clustered organization for a defined community structure. It works as follows:

1. A black-box method is employed to identify the communities within the target network.

2. The communities are isolated and fed into the GP as the target set for the community model population.

3. The inter-community edges form a graph which is fed into the GP as the target for the inter-community model population.

4. Select two models, one evolved from each population, and combine them to create a hierarchical model.

Note that Steps 2 and 3 are performed in parallel. Step 4 is accomplished by using the community model to generate a number of small graphs $\{C_1, .., C_l\}$ that are combined into a graph G_C using a disjoint union. Next a graph, G_O, is generated by the inter-community model using an initially empty edge set, and the vertex set of G_C. The final graph is then constructed $G = \{V(G_C), E(G_C) \cup E(G_O)\}$ where $V(G_C)$, $E(G_C)$, and $E(G_O)$ are the vertex set for G_C, and the edge sets for G_C and G_O respectively. The process of selecting a model from each population for Step 4 will be described in Section 8.3. Figure 8.1 illustrates the steps above.

The community detection algorithm is responsible only for dividing the initial target graph and informing the final model about how many communities it should create. The evolved model is responsible for deciding how it initializes a graph, in what way it adds nodes, how to connect those nodes, and how many edges to do

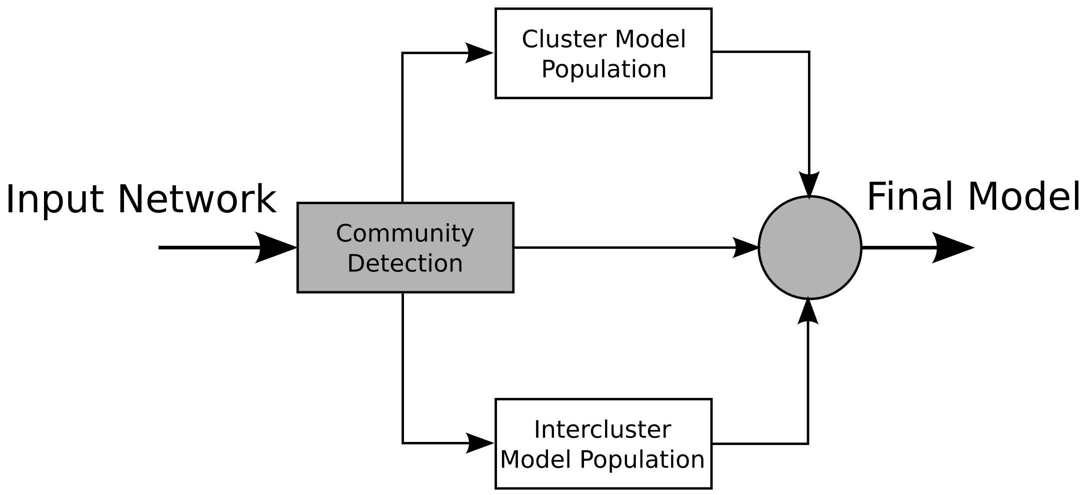


Figure 8.1: The process of generating hierarchical models.

it with. It should also be noted that the community detection algorithm could be exchanged for any method which divides the original network. In this paper, the algorithm used is the leading eigenvector community detection algorithm [71].

There are some issues for further investigation in the proposed method, in particular the number of communities is dependent on a the community detection algorithm external to the GP system. Secondly, the GP system is aware of how the community and inter-community models perform independently, but their interaction is not evaluated. Hence, while a future study will evaluate what the best method for hierarchical model generation is, and further address these issues, this chapter represents the first effort at automatically generating these types of models and our preliminary results already outperform existing models.

8.3 Construction of The Final Model

From the community and inter-community populations, a set of candidate community models, and a set of candidate inter-community models are evolved. Exactly one model from each pool of candidates is needed to construct the final model, in order to select these models from their respective pools, each candidate from each pool was used to create 30 graphs that were compared to their respective target graphs. The comparisons were used to assign a final fitness value to each candidate community and inter-community model with respect to each objective, i. The models were then ranked according to these values. The rankings were then automatically assigned a weight, α_i, according to (7.1) defined in Chapter 7 reproduced here:

$$\alpha_i = \frac{\max(\mu_{ji}) - \min(\mu_{ji})}{\sum_{i=1} \max(\mu_{ji}) - \min(\mu_{ji})}$$

where μ_{ji} is the average fitness objective produced by the j^{th} evolved model. Each evolved model j was then assigned a final weighted rank sum, $RankSum_j$, which was used to automatically select a model from each pool to construct the final model (note that this value is used only for candidate model selection and not as a fitness value during evolution). $RankSum_j$ was defined in (7.2) in Chapter 7, computed as:

$$RankSum_j = \sum_i \alpha_i \times rank_{ji}$$

where $rank_{ji}$ is the rank of the j^{th} individual with respect to the i^{th} objective. Once the ranking process was completed the median, not the best, model was selected to be used for the final hierarchical model. Selection of the best model often led to selection of a model that was both over-fit, and not representative of the quality of the models the GP system was likely to produce. Thus, in selecting the median model the results were found to improve.

8.4 Parameter Tuning

The Barabási-Albert model is primarily concerned with matching the degree distribution of real-world networks [7], and so this the feature that we will focus on tuning. There are two parameters that can be adjusted, the power of the preferential attachment α_{BA} and the number of edges added per iteration, m. By adjusting the power we can influence the shape of the degree distribution, and adjusting m will translate it. The effect of the parameters α_{BA} and m are illustrated in Fig. 8.2 and 8.3. The two parameters were tuned by a linear search of parameter combinations beginning with $m = 1$, and $\alpha_{BM} = 0.1$ and incrementing them by 1 and 0.1 respectively. Fifty graphs were generated with each parameter combination and the tuning process was terminated when the difference between the cumulative average degree distribution histogram from the fifty graphs and the cortex network was minimized. The difference was measured via the KS test statistic and the final values chosen were $\alpha_{BA} = 0.3$ and $m = 11$. This combination produced a KS test statistic of $D = 0.1$ and a *p-value* of 0.9899, indicating this was a very good fit. Fig. 8.4 show how the clustering coefficient and the parameter α_{BA} relate to the value of m, the points are labelled with the values of m they represent. The triangle on the plot shows the values for the clustering coefficient and average geodesic path length of the cortical network for reference. Note that an m value of 15 would have produced graphs with transitivity values more similar to the cortical network. However, an m value of 15 produces a poorer match with respect to the degree distribution, and average geodesic path length.

In tuning the small-world model the value $p_{rew} = 0.08$ was taken from [101] in which the authors were fitting the same model to the same data, and the number of neighbours for the initial lattice was set at 10 to most closely match the number of edges in the cortex network.

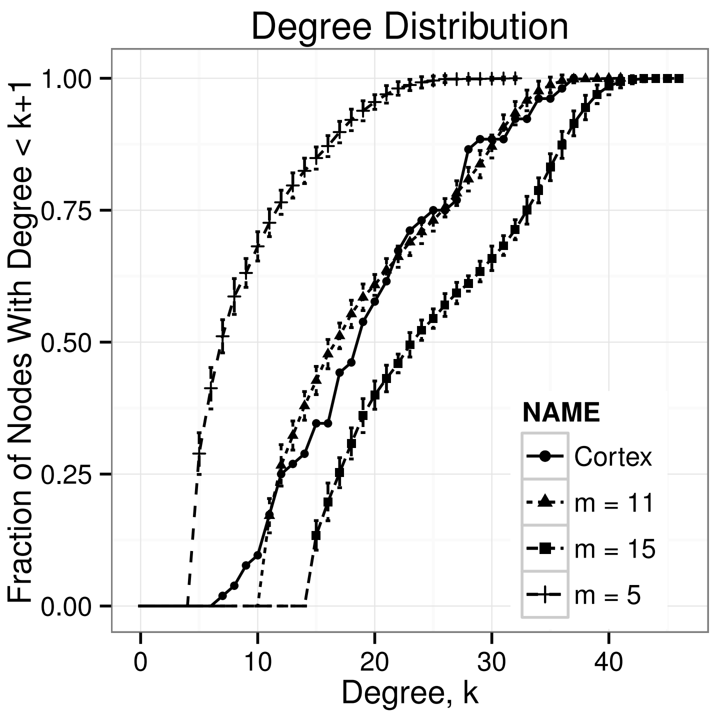


Figure 8.2: The effect of the parameter m on the degree distribution

8.5 Evolving A Cortical Model

A hierarchical model was generated as described in Section 8.2. The parameters used to evolve the model are listed in Table 8.1.

8.5.1 Performance of the Evolved Model

The final model constructs communities by initializing them with a ring of nodes, and then proceeds to create dense interconnections via the `DUPLICATE` function, as well creating some heavily connected hubs with the `CONNECT_STUB_PERSIST` function. It then probabilistically connects all pairs of nodes, and finally rewires a portion of the edges. The inter-community model, responsible for joining the communities, connects nodes by triangles and features hubs created again with a combination of the `DUPLICATE` and `CONNECT_STUB_PERSIST` functions, it also adds some edges probabilistically between all nodes. Finally, it probabilistically removes some edges. The combination of these models produces graphs with dense clusters, prominent hubs, and less dense inter-community connections which are joined with hubs. It also features short-cuts in the form of random edges between communities. The evolved model possess features of both WS model as well as the BA model. The average cumulative degree distributions of 50 graphs produced by each model are plotted against the cumulative degree distribution of the cat cortex in Fig. 8.5. The degree distribution of the WS graphs are clearly dissimilar to the cortical network, while

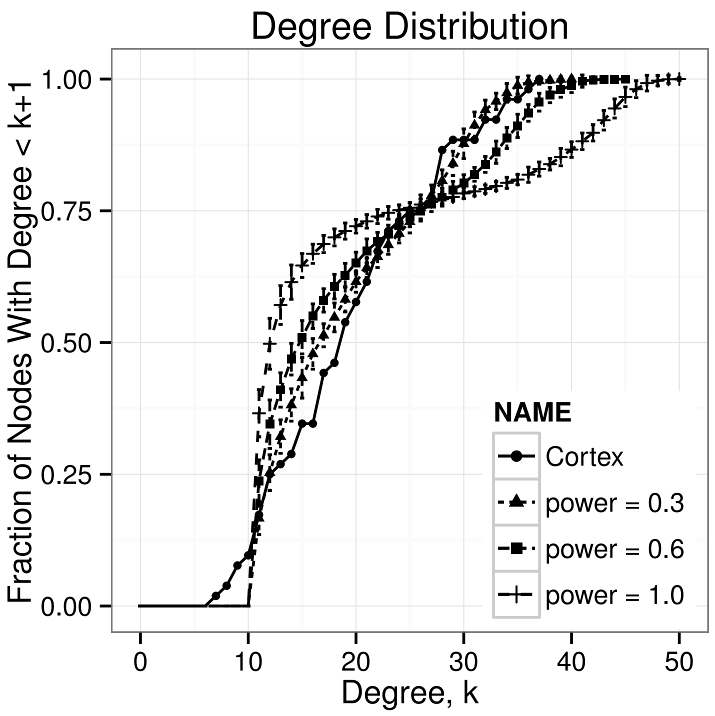


Figure 8.3: The effect of the parameter α_{BA} on the degree distribution

the BA distribution and the evolved model distributions are very similar. A KS test comparing the evolved model's average degree distribution to the cortical network's gives a test statistic of $D = 0.1053$, with *p-value* of 0.9844 which is similar to the fit of the BA model. Even more striking is how the degree distribution of a single evolved graph looks against the cortical degree distribution, as seen in Fig. 8.6. The cortical network distribution has a periodic look to it which is different than the unimodal distributions produced by the BA and WS models. The evolved model is capable of producing these unusual distributions which have reoccurring degree frequencies, similar to the distribution observed in the cortical model.

Tables 8.2, 8.3, and 8.4 show how the models perform with respect to the size of the edge set $|E|$, transitivity (clustering coefficient), diameter, average geodesic path lengths, and the number of communities in the graphs. The far right column shows the squared error between the average values and the values found in the cortical network t. If the error value is bold it means it is at least as small as the smallest error of any of the models. The BA model produces the greatest errors, while the WS model is the closest in terms of edges and is able to consistently match the diameter of the cortical network. However, the evolved model also consistently matches the diameter, and is more similar to the cortical network than the other models with respect to these properties than the other models except the edge count where it falls between the BA and WS models. For reference, the cortical network has $|E_c| = 515$ edges, a transitivity of $C_c = 0.585$, a diameter of 3, an average geodesic path length of $l_c = 1.636$ and 3 communities.

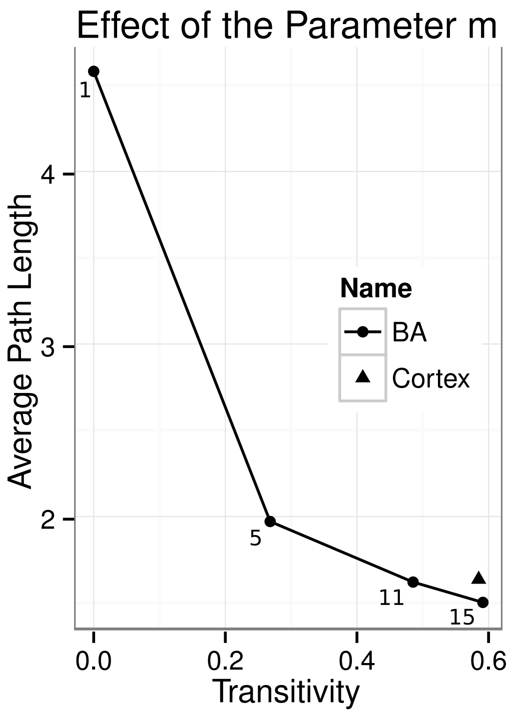


Figure 8.4: The shape of the trees used by the GP

It is thought that the number of alternate shortest paths between any two node pairs (i, j) may have an important impact on information processing within the cortex [101], this value is called the multiplicity, $M_{i,j}$. A good cortical model should be capable of generating similar frequencies of geodesic paths between node pairs. Fig. 8.7 shows how the average frequency of multiplicity values in 50 graphs generated by each model compare to the cortical network. While all three models reasonably reproduce the frequencies of path lengths of one and two, only the evolved model produces a number of paths of length three comparable to those observed in the cortical network. Longer cortical pathways generally travel between different functional areas of the cortex [101], and in the evolved model graphs they travel between different communities. This is a feature the BA and WS models lack, and it is an important feature of the cortical networks given that these longer paths may be responsible for multisensory modulation and integration [101].

Another property which describes how elements of a network communicate is the betweenness centrality, it is affected by community structure and is an important property with respect to cortical networks [35, 90]. The mean betweenness was measured in the fifty graphs generated by each model, and the average of those scores was then taken. The BA and WS models both generate graphs with average betweenness scores more than twice as high as the cortical network, while the evolved model's average betweenness score is within one standard deviation of the cortical network, Table 8.5 lists the results.

Lastly, Figure 8.8, shows the cortical network, and examples of one randomly selected BA, WS and evolved model graph. The node sizes in the plots are propor-

Table 8.1: GP parameters.

Parameter	**Value**
Initialization Method	Koza's 'grow' method
Grow Min	3
Grow Max	5
Maximum Tree Size	17
Population Size	200
Generations	150
Selection	Tournament: k=3
Crossover	Subtree Crossover: 0.95
Mutation	Grow: 0.2, linearly decreasing, $\min \text{depth} = 1, \max \text{depth} = 4$
Runs	50

Table 8.2: Properties of graphs generated by the BA model compared to the cortical network, t.

	Min.	**Q1**	μ	**Q2**	**Max.**	$(\mu$ - $t)^2$
$\|E\|$	506	506	**506**	506	506	$8.10{\cdot}10^{1}$
Trans.	0.47	0.48	**0.49**	0.49	0.50	$9.71{\cdot}10^{-3}$
Diam.	2	3	**2.92**	3	3	$6.40{\cdot}10^{-3}$
Avg. geo.	1.62	1.62	**1.62**	1.62	1.62	$2.48{\cdot}10^{-4}$
Comm.	2	3	**3.52**	4	5	$2.70{\cdot}10^{-1}$

tional to their degree, and the nodes were positioned using the Fruchterman-Reingold algorithm [33]. It is possible to visually distinguish the clustered organization of the cortical network as well as the network generated by the evolved model and how they differ from the BA and WS graphs. The cortex is a complicated structure which possesses an inhomogeneous distribution of connections to other cortical areas, with communities of nodes more densely connected than others. This is reflected in the number and length of communication paths through the cortex. The organization of the communication pathways is important to healthy brain function, and classification and modelling of this behaviour has led to advancements in identifying unhealthy or injured brains. However, the important community structure of the cortex is not modelled by existing algorithms in common use and it is unclear how to properly design or select better algorithms. The results from the previous chapter have shown that GP is a promising method for automatically generating graph models robust to any real-world data, such as cortical networks, especially in the case where good algorithms are unknown. This chapter detailed the first GP system for the automatic inference of graph models capable of generating graphs which exhibit a strong community structure, and it was applied to infer a model of the cat cerebral cortex.

Table 8.3: Properties of graphs generated by the WS model compared to the cortical network, t.

	Min.	**Q1**	μ	**Q2**	**Max.**	$(\mu - t)^2$
$\|E\|$	520	520	**520**	520	520	$\mathbf{2.50} \cdot 10^{1}$
Trans	0.47	0.52	**0.53**	0.54	0.56	$3.22 \cdot 10^{-3}$
Diam	3	3	**3**	3	3	**0**
Avg. geo.	1.61	1.61	**1.62**	1.62	1.62	$4.30 \cdot 10^{-4}$
Comm.	3	3	**3.72**	4	4	$5.18 \cdot 10^{-1}$

Table 8.4: Properties of graphs generated by the evolved model compared to the cortical network, t.

	Min.	**Q1**	μ	**Q2**	**Max.**	$(\mu - t)^2$
$\|E\|$	450	546	**523**	546	546	$6.40 \cdot 10^{1}$
Trans.	0.55	0.55	**0.55**	0.55	0.55	$\mathbf{1.04} \cdot 10^{-3}$
Diam.	3	3	**3**	3	3	**0**
Avg. geo.	1.61	1.61	**1.64**	1.61	1.72	$\mathbf{1.81} \cdot 10^{-5}$
Comm.	3	3	**3**	3	3	**0**

Table 8.5: Betweenness centrality comparison

Model	μ	σ
Evolved	**16.31**	1.22
BA	37.85	0.144
WS	43.03	0.70
Cortex Mean Betweenness = **16.21**		

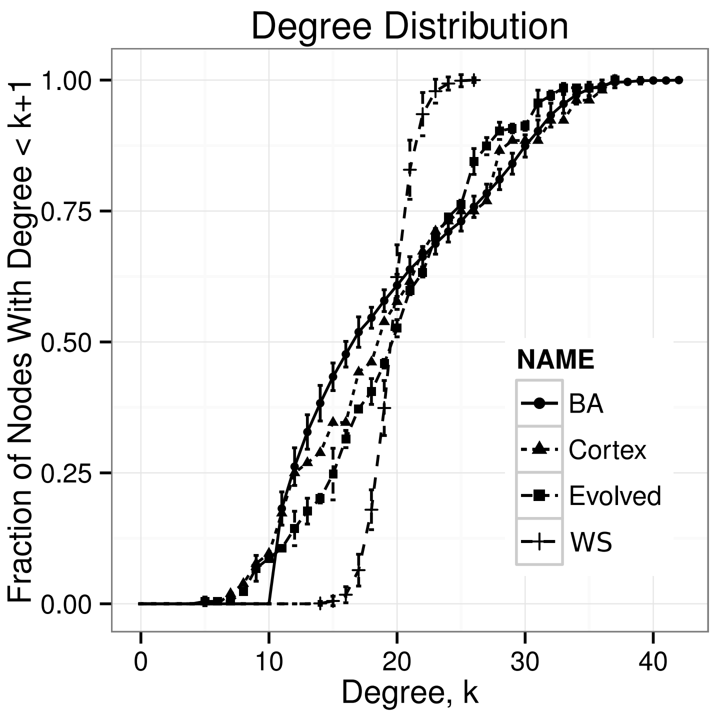


Figure 8.5: Cumulative degree distributions of the models compared to the cortical network

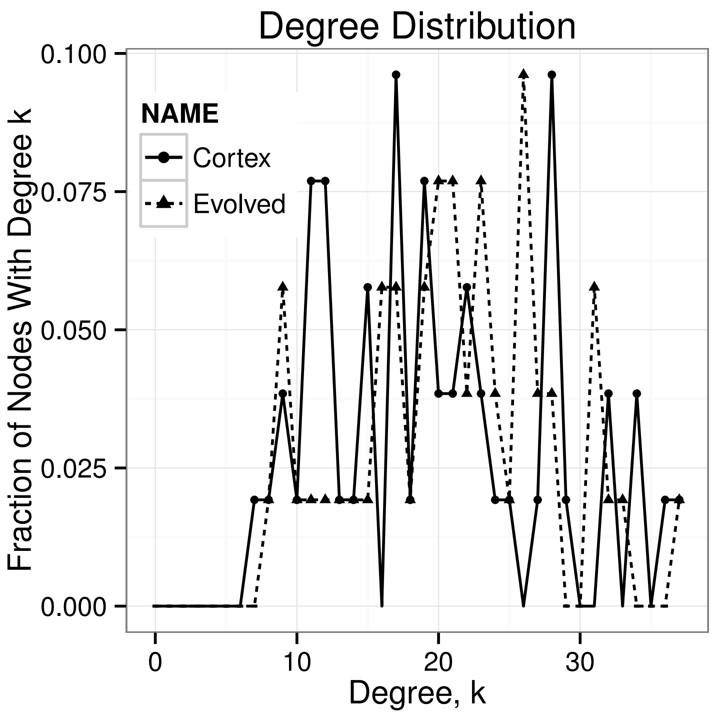


Figure 8.6: A single degree degree distribution produced by the evolved model vs. the cortical network

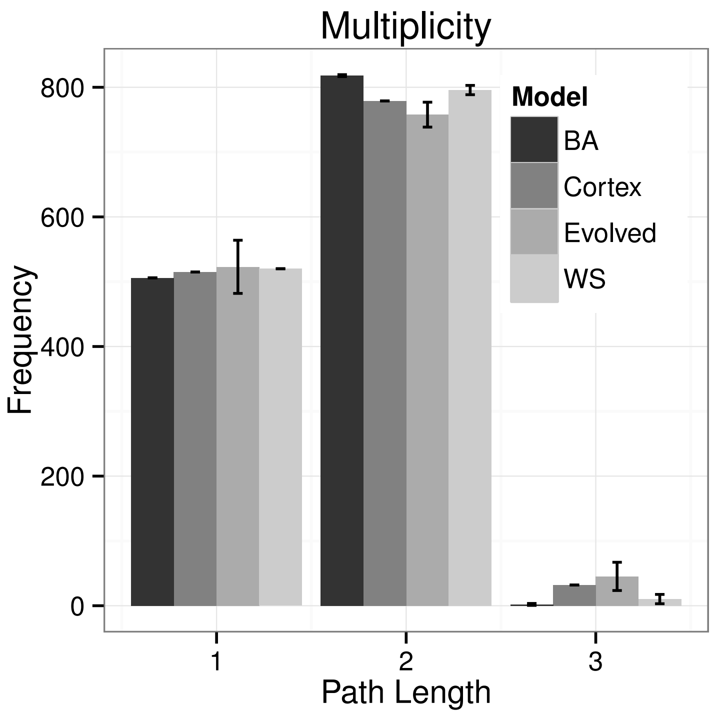


Figure 8.7: The number of shortest paths between all node pairs i, j in all models versus the cortical network.

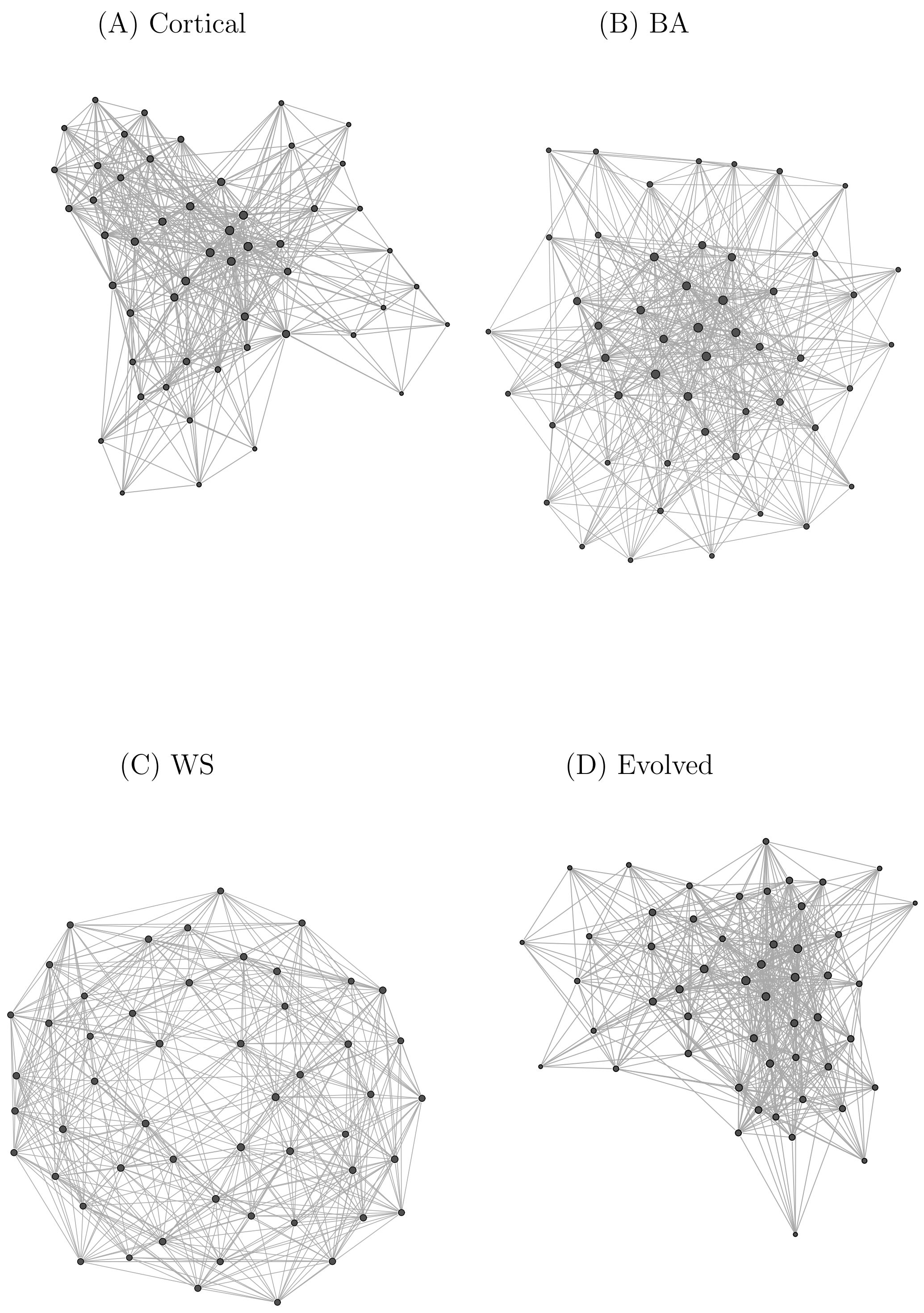


Figure 8.8: The cortical network and a graph generated by each model.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

There have been a number of things that did not come to light until near the conclusion of this research, and from the outset it was not even so clear if these investigations would yield any useful results. The unavailability of prior work in creating a function set for the problem proved to be a formidable obstacle, and some of the early attempts at constructing a useful one were unsuccessful. In reflection, it would have been advantageous to spend a great deal more time on the function set, and in regard to that, it would have been doubly advantageous to construct some proofs as to the behaviour of each function in the limit of large graphs. Though, such an endeavour may still be the focus of future work, it had remained outside the scope of this thesis to the end of showing the application of GP to the problem was at all soluble (which was in itself a considerable effort).

Perhaps equally difficult to establishing a GP function set was the task of deriving a fitness function, or set of functions, capable of guiding the search for useful graph models. The proposed fitness functions have largely been based on empirical evidence as to the usefulness of their related network properties in categorizing different kinds of networks. There has been no work to show the necessity or sufficiency of any one of them, or any set of them, either in the task of categorizing networks or in the context of fitness functions. Exploring the possibility of different fitness functions was briefly considered in various trials of the proposed GP system and they were executed with little success. It is entirely possible that the answer to the question of which properties are optimal lies within the structure of each network considered, and to that end selection of a fitness function may be a combinatorial problem worthy of its own study. The shape of the GP tree was also something derived from intuition and observation of existing models and therefore suffers from the same drawbacks (inability to construct defined community structures, for example). Some modifications to the process used in Chapter 7 were used in Chapter 8 to allow the GP to evolve models for community structure, but the method relies on an external algorithm, further work should consider a more general and expressive structure.

Another important realization that came partway into this journey was that complex networks are truly decentralized in nature and reasoning about connectivity should come almost exclusively from local properties of vertices (or perhaps from a "nearby" area). In some early trials, various node properties (such as centrality) were included as terminals in the GP language and were sometimes effectively used by the GP system, but largely they were used ineffectively as a way to generate floating point values, etc. Obviously, the ineffective use of such functions made the system unusable on larger input graphs and initially the decision to remove them from the function set was resisted because they seemed important, at least ostensibly. However, removing them had little impact on the system's ability to yield useful results, corroborating the feeling that connectivity should not depend on more information than what can be determined locally.

It seems plausible that local connectivity rules could be responsible for many network properties (cumulative advantage, transitivity, etc.), but what about community structure? There was no obvious answer to this question uncovered in the short time during this research that community structure was "under the microscope," so to speak. Community structure was never intended to be an object of study here, but the preliminary examinations of cortical networks and other real-world networks have suggested that it is perhaps one of the most powerful factors in the organization of real networks. Further studies would do well to determine how community structure can be derived spontaneously without the need for considering the global importance of vertices.

The challenges of this work did not begin or end at the theoretical, a challenge of significant proportion was computational. The complexity of many graph algorithms make them prohibitive to execute on large graphs, and in particular an evolutionary system may execute them many times depending on their use in the system (as a GP language function perhaps). This line of research would naturally be complemented by the derivation of fast graph algorithms, whether they be methods for computing network properties or estimates of those properties, or methods of fast graph sampling.

This research has contributed the proposal of the first system for the automatic inference of graph models for complex networks. It has showed that GP is a capable tool for this problem, and laid the groundwork for future research by proposing a GP function set and fitness function able to create useful and expressive graph models. It was also shown that the proposed system could be used to infer models for real systems by evolving a model for the cortical network of a cat, which was more accurate than standard models. Finally, the cortical model also showed that evolving models for community structure was possible.

9.2 Future Work

This thesis has discussed only undirected and unweighted networks, and briefly touched upon the issue of community structure. Many natural and artificial systems are actually best represented by directed or weighted networks (or both). For example, social networks may best be represented as directed, because friendship may not be reciprocal, and weighted, because some people may be better friends or more immediately influential than others. Furthermore, because of the complications that arise from dealing with large datasets this work has focused on relatively small networks (hundreds of nodes at most). However, given the size of many real-world networks it may be more realistic to use much larger datasets when deriving a model. The following subsections will briefly discuss these possible future research directions.

9.2.1 Directed and Weighted Networks

There is no obvious way to extend the GP methodology discussed in this thesis for directed networks. The function set specific to directed networks would need to be derived. Although, it is possible that the shape of the GP tree could be adapted and that the functions in a directed language could be similar in some ways. Furthermore, a fitness function or functions suitable for use in evolving graph models of directed networks would need to account for both the in and out degree distributions in a directed network.

Weighted networks would again present the challenge of designing a new function set, as an appropriate mechanism for creating edge weights would need to be present. Additionally, at the very minimum a fitness function would be required to measure differences in weight distributions between the active and target networks.

9.2.2 Models of Community Structure

It is presently unclear how to best tackle the challenge of evolving models with naturally occurring community structure. The mechanisms that cause communities to appear and grow likely depend on dynamic network properties. Therefore, a methodology for inferring models of community structure would need access to, or be able to synthesize, information about how properties that influence communities change over time. For example, one study proposes "trendiness" as a function of node degree and time [60]. Identifying the features that encourage community structure and unifying their properties would itself be a significant contribution to the field of network science.

9.2.3 Large Networks

Networks such as the Internet, WWW, protein-protein interaction networks, and networks of neural connectivity in the brain are enormous and could contain millions or billions of nodes. Furthermore, these networks are of great importance to understanding our world. It may be possible to infer useful models from small subsets of these networks, but gathering a representative subset is sometimes in itself a challenge. In some cases, there is network data over time. These kinds of networks are perfect candidates for automatic model inference as a sequence of small networks can be used to infer a model capable of generating an enormous network.

If network growth data is not available the network may need to be sampled. However, generating representative network samples is non-trivial, and has been a topic gathering increasing interest in the social science community [64]. If the network cannot reasonably be sampled the methodology proposed here could potentially still be used with some modifications. Fortunately, GP is a so-called "embarrassingly parallel" algorithm [77] so the process of selection, crossover and mutation, and evaluation is simple to scale for many CPUs or distributed architectures. Additionally, it is possible to compute many graph properties using massively parallel systems designed for processing huge graphs of millions or billions of vertices [57]. Access to such a system would obviously be advantageous for serious research in the area of complex networks.

Appendix A

Early Experiments: Establishing a Function Set and Fitness Objectives

This appendix shows some abbreviated results from an earlier iteration of the GP system outlined in Chapter 6, which may be of some interest because they highlight some successes and failures of an earlier GP function set. More importantly, these results led to some important improvements to the GP language, specifically the creation of the `CONNECT_STUB_PERSIST` function and the addition of the Sum of Percent Differences fitness objective.

GP Language

This particular early iteration of the GP system outlined in Chapter 6 shared with it the same GP tree structure, and much of the function set was identical. However, it lacked the `DUPLICATE` and `CONNECT_STUB_PERSIST` functions. The function set also included some now defunct operators, such as terminals to compute a node's betweenness and closeness centrality.

The function set of the old system is given below without descriptions, except where they vary from those described in Chapter 6. In addition to a basic set of math operators, $\{+, -, *, \%\}$, Ephemeral random constants (ERC), IF structures, and the relational $<$ operator, the language includes:

- Initialization actions:
  - `ADD_ALL_NODES`
  - `BUILD_RING`
  - `SET_GROW_NODES`
- Growth actions:
  - `CREATE_TRIANGLE`
  - `CONNECT_W_PROB(p)`
  - `CONNECT_RAND`
  - `CONNECT_STUB(p)`
- Finalization actions:
  - `REWIRE_EQUAL_PROB(p)`
  - `REWIRE_RANDOM`
  - `REMOVE_PROB(p)`
- Terminals:
  - `CURRENT_NODE_DEGREE`
  - `AVG_DEGREE`
  - `MAX_DEGREE`
  - `BETWEENESS_CENTRALITY`: Gives the betweenness centrality of the current vertex in the active graph.
  - `CLOSENESS_CENTRALITY`: Gives the closeness centrality of the current vertex in the active graph.
  - `TOTAL_VERTEX_COUNT`: Returns the vertex count of the active graph.
  - `TOTAL_EDGE_COUNT`: Returns the edge count of the active graph.

Probabilities are automatically selected from a pre-generated list, P of probabilities where,

$$P = \{0.01, 0.02, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}.$$

All functions that take probabilities as an argument take a floating point number, a, as a parameter. The value a is mapped to an index, $i = a \text{ MOD } |P|$. The probability $p = P_i$ is then passed to the function.

Fitness Function and Evaluation

The fitness functions defined for the old system were identical to those given in Chapter 6, except that the Sum of Percent Differences (SPD) function had not yet been included. Weighted and normalized summed-ranks was used as the multi-objective strategy with manual weight selection. The weights were empirically established and set at $\frac{1}{4}$, $\frac{1}{4}$, $\frac{1}{2}$, for F_1Raw, F_2Raw, and F_3Raw, respectively. It was found via preliminary experimentation that the KS-test statistic was much harder to minimize than the average path length or clustering coefficient.

Objective weighting was the simplest way to keep the GP from trading off small gains in the differences in average path length and clustering coefficient at the expense of the fit of the degree distribution. The decision for using a squared difference versus an absolute difference was established empirically for F_1Raw and F_2Raw.

Active graphs were simplified as described in Section 6.2 before fitness values were computed, and any edge cases (no paths between vertex pairs, etc.) were also handled similarly.

Experimentation and Results

Experiments were conducted with the intent to see if the GP system was capable of reproducing good approximations of the ER, BA, and WS models (the same as the experiments discussed in Chapter 7). The target graphs used as input were generated using the same parameters as those in Chapter 7. However, some other parameters and are shown in Table A.1 and some methods were changed as well (more input graphs, etc.) and the experiments are described below.

Table A.1: GP parameters.

Parameter	**Value**
Generations	70
Population Size	50
Initialization Method	Grow
Grow Min	3
Grow Max	5
Maximum Tree Size	17
Selection	Tournament, k=3
Graphs per evaluation	3
Crossover	Subtree Crossover, 0.90
Mutation	Grow, 0.1, linearly decreasing
	min depth $= 1$, max depth $= 4$
Runs	30

In the first experiment the GP system was initially used to evolve three populations of algorithms meant to approximate the Erdös-Rényi model. Data was collected using 200, 300, and 400 vertex target graphs, respectively. The second experiment was meant to evolve individuals which approximated the Barabási-Albert model, a 200 vertex graph generated by the Barabási-Albert model was used as a target. The third experiment evolves individuals to approximate the Small-world model, a 200 vertex graph was used as a target. The best individuals from each experiment were collected by comparing their fitness values, and by inspection of the average degree distribution plot vs the degree distribution plot of the target models. The best of these evolved algorithms were called ERo_{200}, ERo_{300}, ERo_{400}, SWo_{200}, and BAo_{200}. The first two letters of their names indicate the type of target model which was used to evolve the individual, and the numeric part indicates the size of that target model.

The fourth and final experiment, was conducted in order to examine how well the evolved models predicted the growth of their target graphs. Once the individuals are collected, they are each used to generate thirty graphs at sizes of 200, 400, 600, 800,

Algorithm 4 ERo_{200} algorithm

```
$P[] = \{0.01, 0.02, 0.05, 0.1,$
$0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$;
BUILD_RING(); {Add all nodes and connect in a ring.}
for each node $n$ do
  CONNECT_W_PROB($P$[TOTAL_EDGE_COUNT() $MOD$ 13]);
end for
```

Algorithm 5 ERo_{300} algorithm

```
ADD_ALL_NODES();
for $n$ in $1..N$ do
  CONNECT_W_PROB(0.01);
  CONNECT_RAND(); {Connect node $n$ to a random node.}
  CONNECT_STUB(TOTAL_VERTEX_COUNT()); {Satisfy an edge request in the first
  $P[|V|] * |queue|$ portion of the request queue or make a request if there are none
  to satisfy.}
  CONNECT_W_PROB(0.01);
end for
REWIRE_RANDOM(); {Randomly rewire all edges.}
```

and 1000 vertices. Graphs of the same number of vertices were then generated by the Erdös-Rényi, Barabási-Albert, and Small-world models. The graphs produced by ERo_{200}, ERo_{300}, ERo_{400}, SWo_{200}, and BAo_{200} were compared to graphs of the same size produced by the model they were to approximate. The fitness functions were applied in order to compare the graphs produced by the evolved models to the graphs produced by the original algorithms. The average fitness values, μ, and the standard deviations, σ, were recorded. Being able to predict future trends with the model is enormously important and informative, and it is a major motivation for this work. That is, producing only graphs of the same size as the target is of very limited use and we show here the ability of the evolved graphs to capture the growth dynamics of the given algorithms.

Results

The ERo_{200} algorithm in Algorithm 4 bears a strong similarity to the Erdös-Rényi model given in Chapter 4. It makes use of the CONNECT_W_PROB function. However, the BUILD_RING function adds some additional structure, guaranteeing a connected graph exists, the Erdös-Rényi model makes no such guarantee. However, the target graph does contain a very large connected component which seems to have led to this outcome.

The ERo_{300} algorithm is more similar to the Erdös-Rényi model, as shown in Algorithm 5. It adds all vertices to the graph, then systematically adds edges. Note that in the case of a 300 node graph the probabilistic section adds $nodes_p = 0.01 *$

Algorithm 6 ERo_{400} algorithm

```
$P[] = \{0.01, 0.02, 0.05, 0.1,$
        $0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$;
ADD_ALL_NODES();
for $n$ in $1..N$ do
  CONNECT_W_PROB($P$[TOTAL_EDGE_COUNT() $MOD$ 13]);
  CONNECT_STUB(CLOSENESS_CENTRALITY() $MOD$ 13);
  CONNECT_W_PROB($P$[TOTAL_EDGE_COUNT() $MOD$ 13]);
  CONNECT_RAND();
  CONNECT_STUB(BETWEENNESS_CENTRALITY() $MOD$ 13);
end for
REWIRE_RANDOM();
```

$300 * 2 * 300 = 1800$ nodes. The CONNECT_STUB function adds 150 edges – one edge for each vertex pair. This gives a total of $1800 + 150 = 1950$ edges. The edges are then rewired uniformly over the entire graph. This is similar to the idea of adding a set number of edges randomly to a graph which contains only vertices. In fact, the algorithm automatically discovered a different formulation of the Erdös and Rényi model. The probability of adding an edge in the evolved model becomes [74]:

$$p = m \begin{pmatrix} n \\ 2 \end{pmatrix}^{-1} = 1950 \begin{pmatrix} 300 \\ 2 \end{pmatrix}^{-1} = 0.0435, \tag{A.1}$$

which is very close to the probability parameter used to construct the target graph.

The ERo_{400} model in Algorithm 6 also uses ADD_ALL_NODES initialization function. Note the use of the REWIRE_RANDOM function, which destroys any initial structure to the graph. The sum result of these operations is to assure the correct number of edges are in the model – similar to the behaviour observed in ERo_{300}.

Algorithm 7 BAo_{200} algorithm

```
SET_GROW_NODES(); {Create an empty graph, add a new node at each iteration.}
for $n$ in $1..N$ do
  CONNECT_RAND();
end for
```

The BAo_{200} model in Algorithm 7 utilizes the GROW_NODES function, the Barabási-Albert algorithm also adds nodes on each iteration as opposed to all at once. Although the BAo_{200} algorithm shows no solid mechanism for preferential attachment. Using GROW_NODES in conjunction with the CONNECT_RAND function means that nodes added earlier on in the construction of the graph will have higher opportunity to gather edges. The Barabási-Albert model produces many nodes of low degree and only a small number of nodes of high degree, and thus the evolved model is capturing some of this preferential attachment behaviour. However, objective F_3 will not heavily

Algorithm 8 SWo_{200} algorithm

```
BUILD_RING();
for $n$ in $1..N$ do
  CONNECT_RAND();
  CREATE_TRIANGLE();
  CREATE_TRIANGLE();
end for
REMOVE_PROB(TOTAL_VERTEX_COUNT() $MOD$ 13);
REMOVE_PROB(TOTAL_VERTEX_COUNT() $MOD$ 13);
```

penalize models which do not produce node degrees that occur with low frequency. Future research will propose methods capable of addressing this issue.

The SWo_{200} model, shown in Algorithm 8 is strikingly similar to the actual Small-world algorithm which produced its target graph. The SWo_{200} algorithm creates a ring, creates two triangles – the connection pattern which contributes to a high clustering coefficient – and adds one random edge, creating shortcuts across the ring and a short average path length. It finally removes some number of edges probabilistically, breaking up some of the structure introduced by the triangles. A similar effect to rewiring some edges with equal probability as occurs in the target model.

Figures A.1, A.2 show comparisons of the average histogram produced by thirty 1000 vertex evolved graphs per model to a target graph of the same size of the corresponding target algorithm. The error bars represent one standard deviation. The plots show that the distributions are approximately the same shape and that they overlap considerably. Figure A.3 shows a 200 node graph generated by the BAo_{200} evolved model and the corresponding target graph produced by the Barabási-Albert model is shown in Figure A.4. They display a similar branching structure, but the graph produced by the evolved model does not contain a similar number of high degree vertices. As previously indicated, this will be resolved in subsequent research.

Table A.2 shows the average results, μ, and the standard deviations, σ, of the comparisons of graphs produced by the evolved models to target graphs of various sizes which were produced by the algorithms the individuals were evolved to emulate. A fitness value of 1 is the best possible, and 0 is the worst.

Also, Table A.2 shows how the evolved models performed when they were used to generate graphs much larger than the initial target graphs. Each evolved model generated thirty graphs of each size and the graphs they produced were compared to target graphs of the same size using fitness functions F_1, F_2, and F_3. The averages and standard deviations from the comparisons are shown in the table. Each model does very well at each target size, and the standard deviations show there is very little variation in the kinds of graphs the evolved models produce.

Three problems with the old system were evident from these results. First, that the fitness functions were not sensitive enough to tailed degree distributions (this was apparent from the SWo_{200} model's propensity to generate a long tail, and the failure of the BAo_{200} to generate a tail long enough coupled with the fact that the fitness ob-

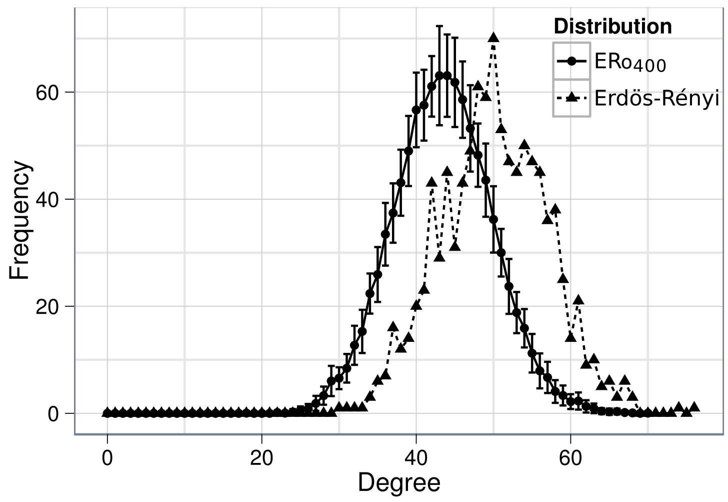


Figure A.1: The average ERo_{400} histogram of thirty 1000 vertex graphs to the histogram of a 1000 node Erdös-Rényi graph.

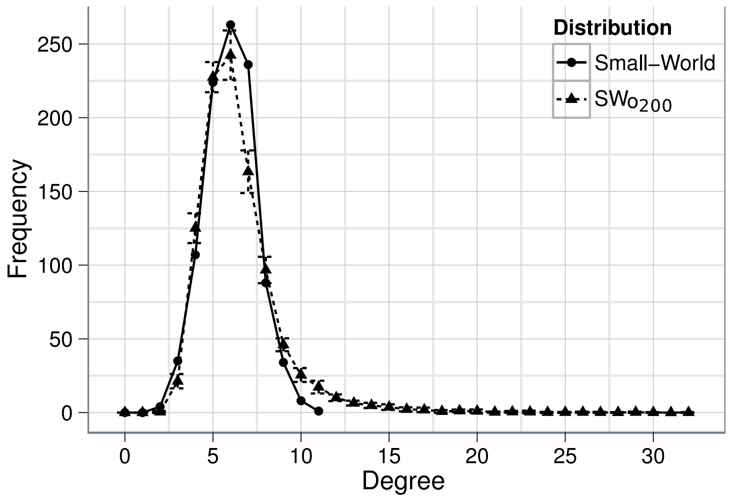


Figure A.2: The average SWo_{200} histogram of thirty 1000 vertex graphs to the histogram of a 1000 node Small-world graph.

Table A.2: Comparison of graphs produced by evolved models to graphs produced by real algorithms

Model	Obj	Size, n									
		$n = 200$		$n = 400$		$n = 600$		$n = 800$		$n = 1000$	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
ERo_{200}	F_1	0.997	7.34×10^{-5}	0.999	2.02×10^{-5}	0.999	2.65×10^{-5}	0.999	8.75×10^{-6}	0.999	5.01×10^{-6}
	F_2	0.978	2.93×10^{-3}	0.973	1.22×10^{-3}	0.974	8.32×10^{-4}	0.974	7.98×10^{-4}	0.973	6.41×10^{-4}
	F_3	0.810	0	0.844	1.26×10^{-3}	0.875	6.42×10^{-3}	0.889	1.11×10^{-2}	0.898	5.28×10^{-3}
ERo_{300}	F_1	0.997	1.91×10^{-4}	0.999	3.41×10^{-5}	0.999	1.35×10^{-5}	0.999	6.19×10^{-6}	0.999	6.03×10^{-6}
	F_2	0.986	6.56×10^{-3}	0.975	1.23×10^{-3}	0.975	1.24×10^{-3}	0.975	7.33×10^{-4}	0.974	4.76×10^{-4}
	F_3	0.857	9.80×10^{-3}	0.867	1.12×10^{-2}	0.883	6.98×10^{-3}	0.894	6.85×10^{-3}	0.903	6.23×10^{-3}
ERo_{400}	F_1	0.999	1.02×10^{-4}	0.999	2.95×10^{-5}	0.999	1.48×10^{-5}	0.999	7.72×10^{-6}	0.999	4.61×10^{-6}
	F_2	0.993	3.38×10^{-3}	0.997	1.34×10^{-3}	0.996	9.08×10^{-4}	0.995	6.71×10^{-4}	0.994	3.89×10^{-4}
	F_3	0.928	1.80×10^{-3}	0.964	7.51×10^{-3}	0.961	7.44×10^{-3}	0.960	4.85×10^{-3}	0.955	4.53×10^{-3}
BAo_{200}	F_1	0.988	0	0.991	0	0.995	2.28×10^{-4}	0.995	8.53×10^{-5}	0.998	8.14×10^{-5}
	F_2	1	0	1	0	1	0	1	0	1	0
	F_3	0.95	0	0.81	0	0.860	8.03×10^{-3}	0.826	1.75×10^{-3}	0.832	1.01×10^{-2}
SWo_{200}	F_1	0.998	1.63×10^{-4}	0.999	8.34×10^{-5}	0.999	3.78×10^{-5}	0.999	2.28×10^{-5}	0.999	2.04×10^{-5}
	F_2	0.957	5.07×10^{-3}	0.949	2.99×10^{-3}	0.968	2.59×10^{-3}	0.952	1.75×10^{-3}	0.971	1.49×10^{-3}
	F_3	0.932	2.28×10^{-2}	0.920	1.18×10^{-2}	0.949	1.33×10^{-2}	0.956	9.12×10^{-3}	0.927	1.50×10^{-2}

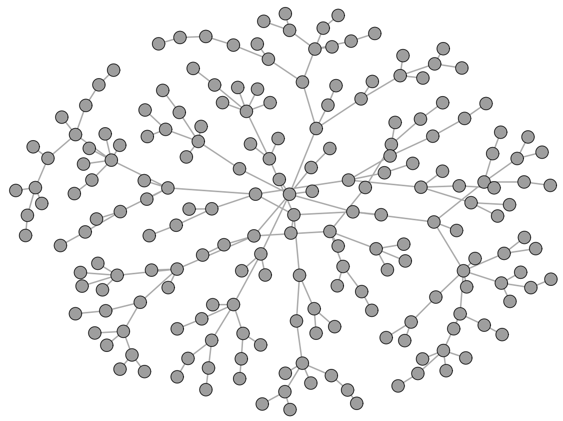


Figure A.3: A 200 node graph produced by the BAo_{200} model.

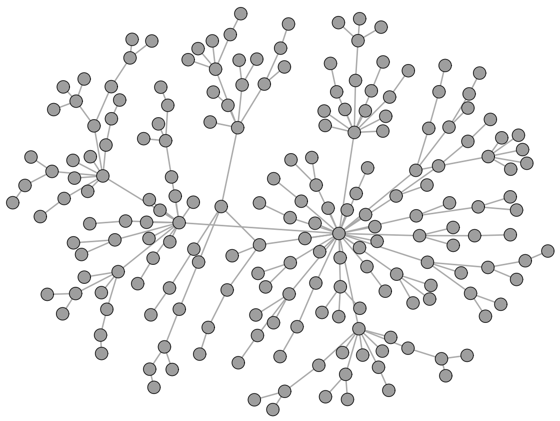


Figure A.4: A 200 node graph produced by the Barabási-Albert model.

jectives never indicated a poor fit). What was needed was a fitness objective sensitive to relative differences in degree frequency, and thus the SPD objective was devised. Secondly, upon further experimentation it became obvious that the language was incapable of generating scale-free behaviour, and the closest it could come to it was to use the `GROW_NODES` function combined with random connections. However, this kind of model has been considered by others and has been shown incapable of generating scale-free behaviour [7]. The solution to the second problem was to introduce a function able to create scale-free behaviour, and thus the `CONNECT_STUB_PERSIST` function was created (although its expressive power seems to not be limited to generating scale-free degree distributions). Lastly, there was no method by which to select a final model from the candidates and in this iteration of the system it was a manual process, the problem was eventually rectified by using the model selection method outlined in Chapter 6.

Appendix B

Reproducing Existing Models: Supplementary Results

These are supplementary results to those presented in Chapter 7. The first section of this appendix contains samples of graphs generated by the evolved models, the section section contains the convergence chart for the experiments, and the third section contains the actual programs output from the GP system which were used to generate the results.

Graph Figures

Fig. B.1 is the Erdös-Rényi graph which was used as a target for the GP system. Fig. B.2 is a graph produced by the evolved model ER_{200}. Both graphs are very dense, and from the node sizes it can be observed that the node degrees are roughly equal throughout and between the two graphs. There is no clear tree-like structure or important hubs in these graphs, as should be expected from a random graph.

The Watts-Strogatz model was used to produce the graph shown in Fig. B.3. The evolved model, SW_{200}, was used to generate the graph shown in Fig. B.4. Note that the graphs lack any obvious hubs or branches, and ostensibly look similar to the Erdös-Rényi graph in Fig. B.1. However, these graphs are much less dense than the Erdös-Rényi graph and the node degrees are noticeably smaller. It is also possible to pick out triangles around the edges of Fig. B.3 and Fig. B.4, whereas the Erdös-Rényi graph has no apparent triangles.

The Barabási-Albert target graph, is shown in Fig. B.5. It is sparse, and has an easily discernible tree-like structure with obvious hubs. The evolved model, BA_{200}, was used to generate the graph shown in Fig. B.6. Fig. B.7 and Fig. B.8 show the 800 node target used for growth experiments and an 800 node graph generated by BA_{200} respectively.

GP Convergence

Figures B.9, B.10, and B.11 show the average per generation behaviour of the absolute fitness objectives (0 is the worst value, 1 is the best) over 50 runs for the ER_{200}, SW_{200}, and BA_{200} experiments respectively. The difference in average geodesic path length (GPD), and the difference in clustering coefficients (GTD) is effectively zero throughout the run while the difference in KS test statistics (KS) and Sum of Percent Differences (SPD) are more slow to change – Matching the degree distribution of the

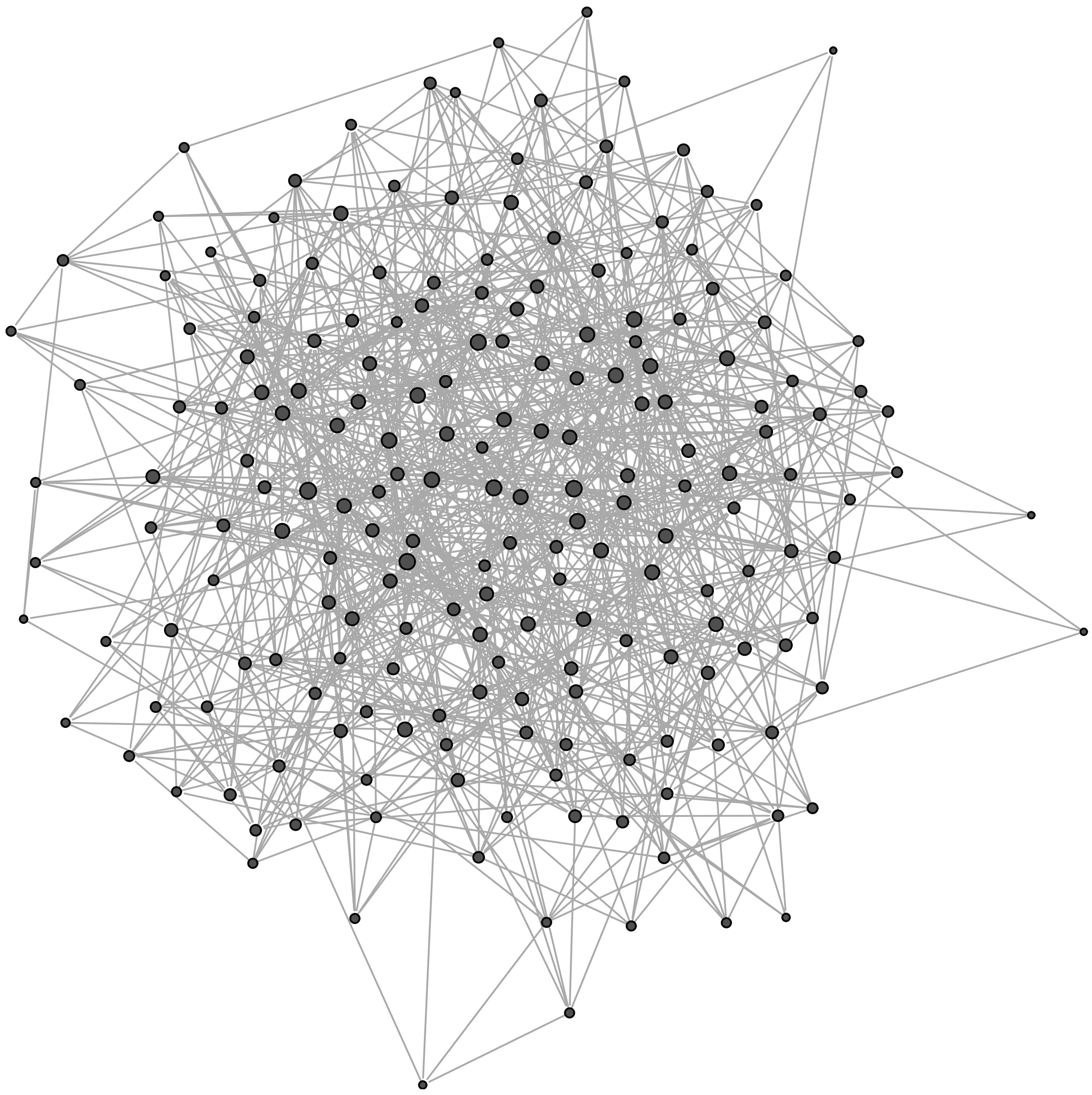


Figure B.1: Erdös-Rényi target, $n = 200$

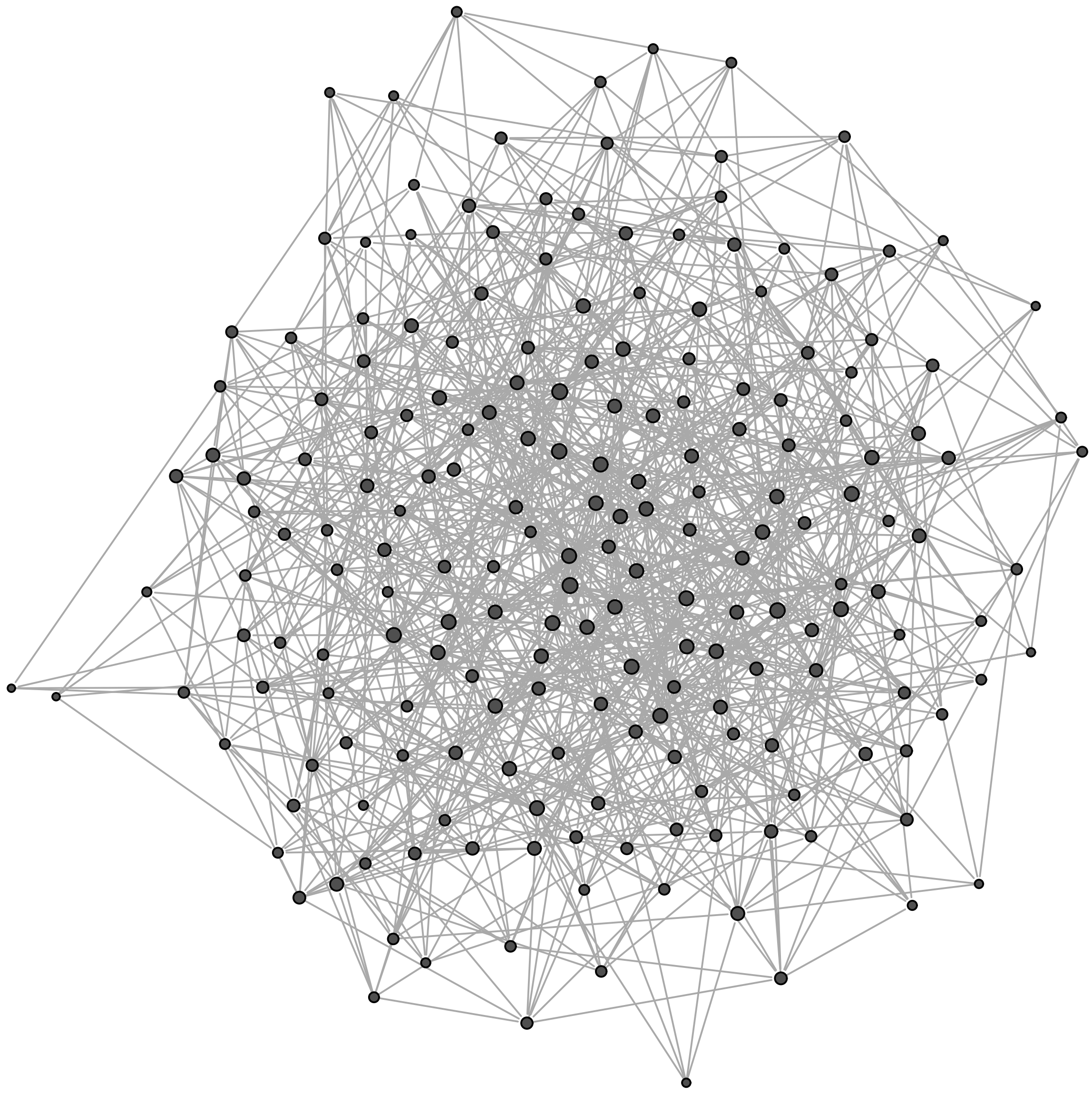


Figure B.2: ER_{200} graph, $n = 200$

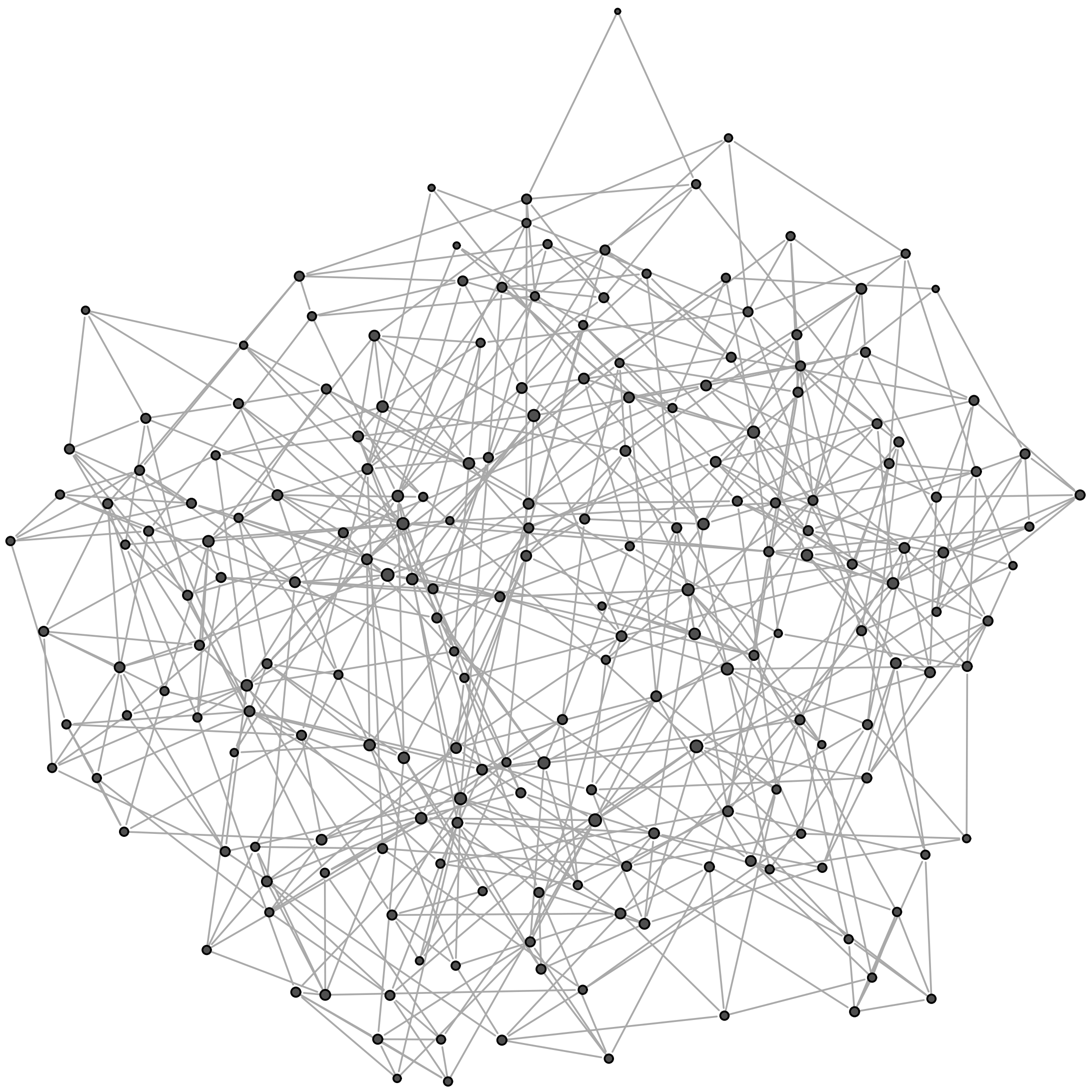


Figure B.3: Small-world target, $n = 200$

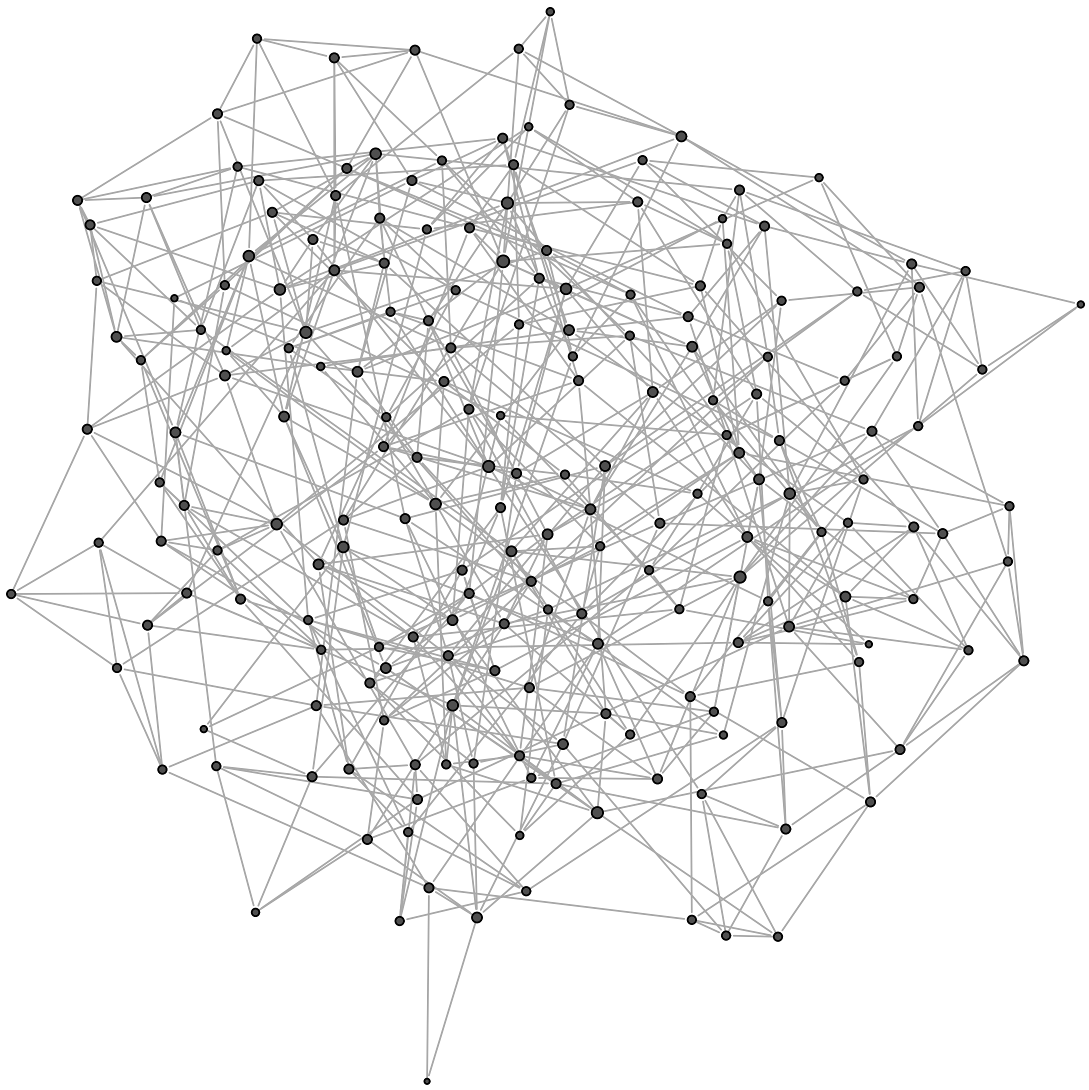


Figure B.4: SW_{200} graph, $n = 200$

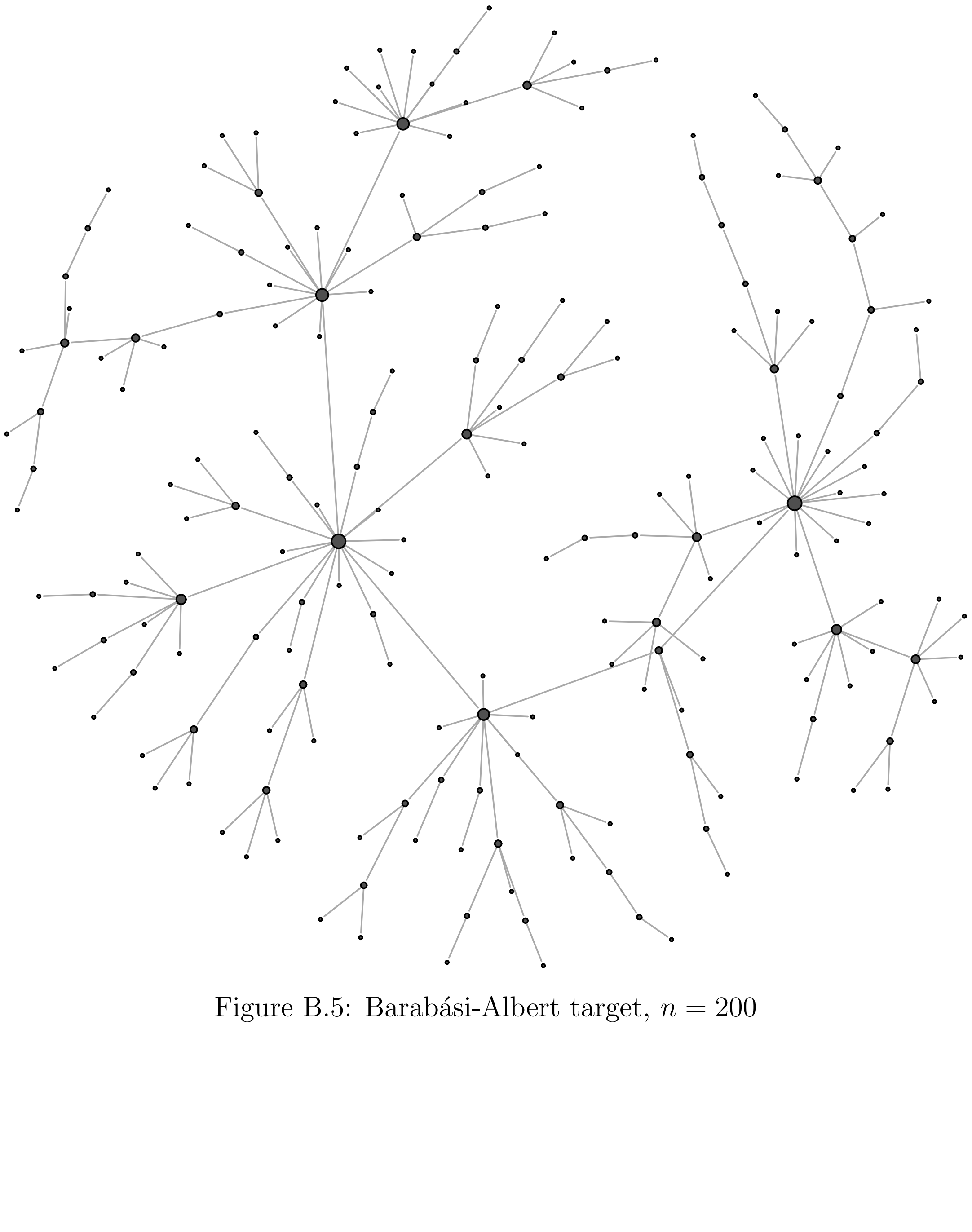


Figure B.5: Barabási-Albert target, $n = 200$

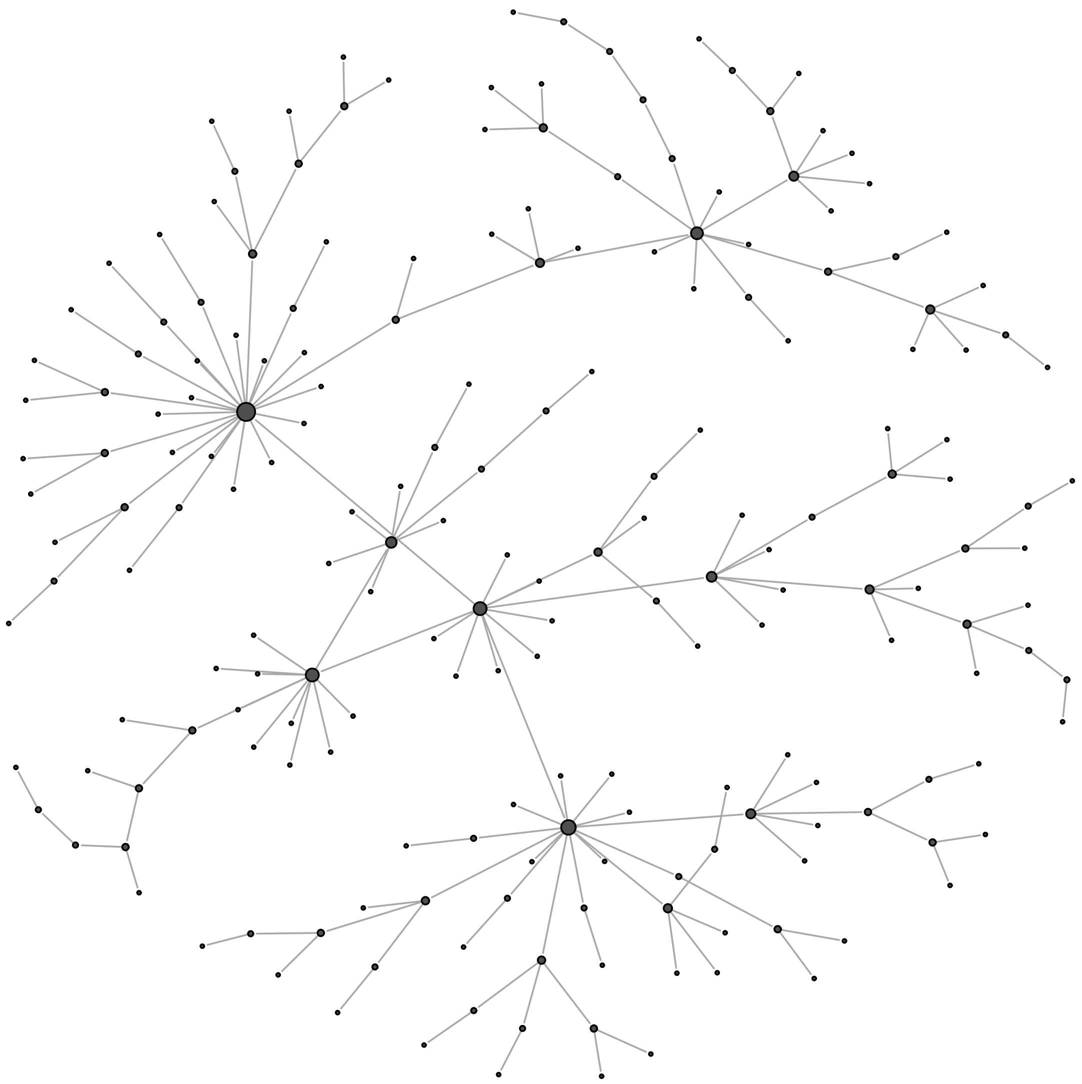


Figure B.6: BA_{200}, $n = 200$

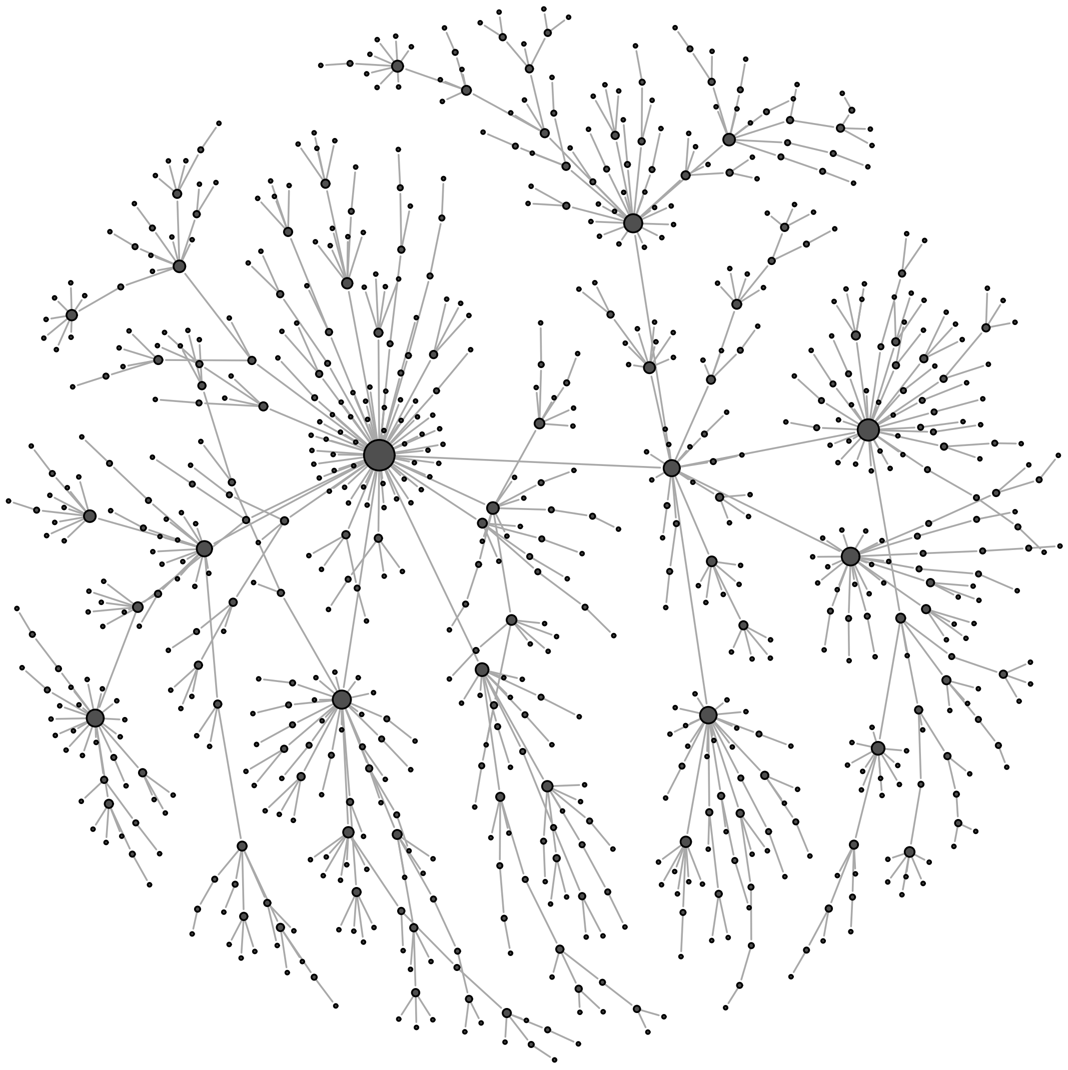


Figure B.7: Barabási-Albert target, $n = 800$

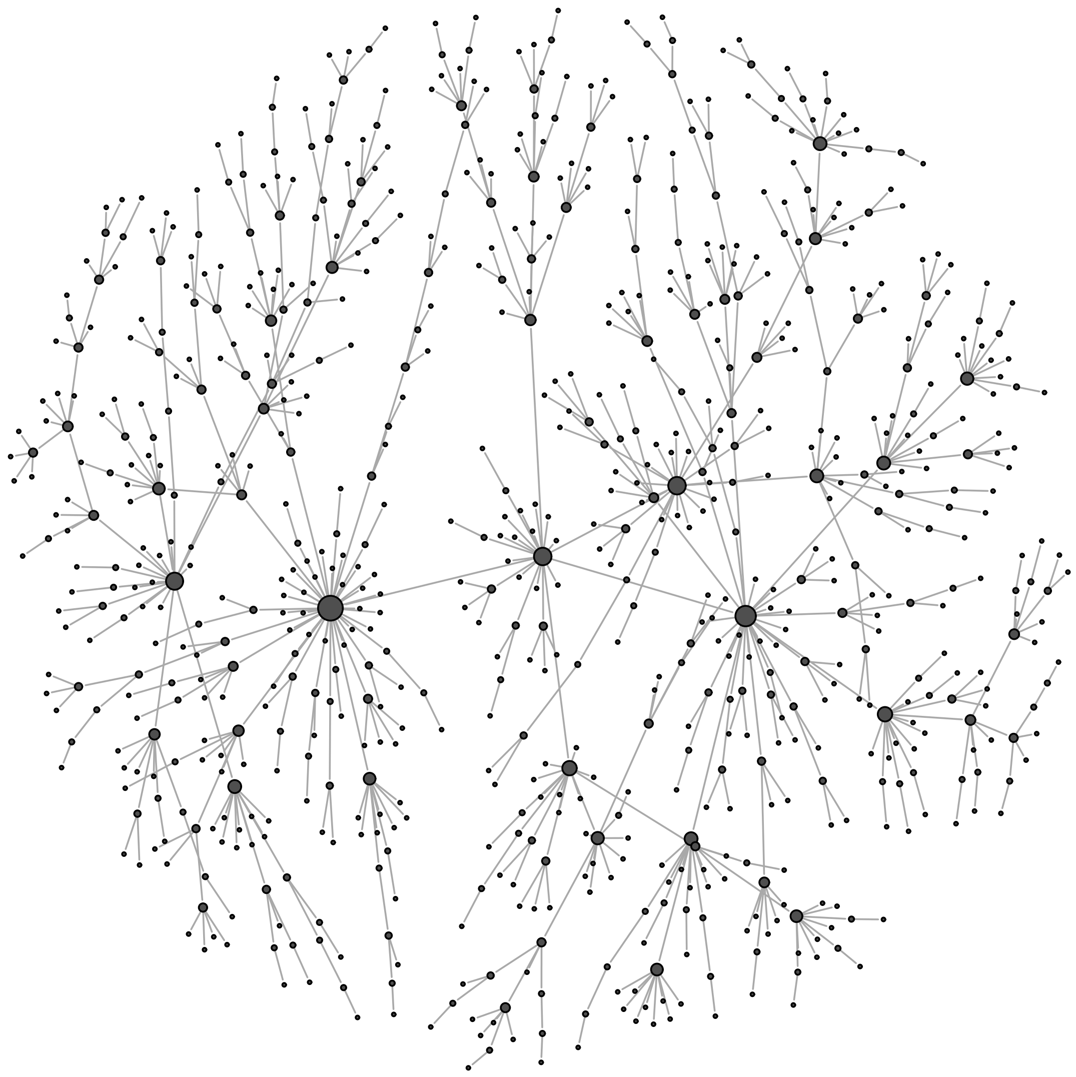


Figure B.8: BA_{200}, $n = 800$

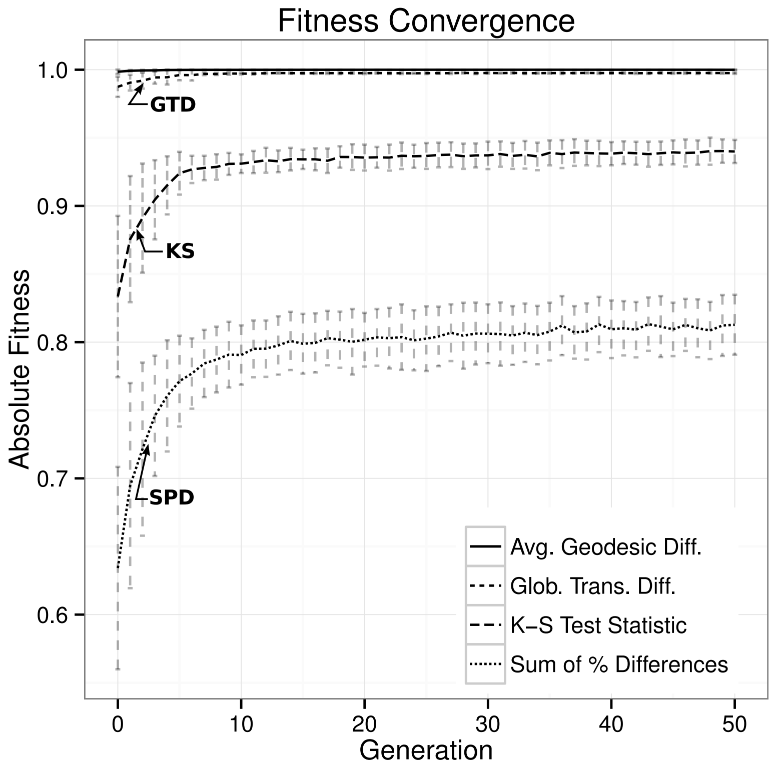


Figure B.9: Erdös-Rényi experiment. Average fitness convergence over 50 runs.

target is more difficult than matching the average path length value or the global transitivity values. However, without the presence of the GPD or GTD objectives the quality of the final results may suffer.

GP Trees

This section contains the actual output from the GP system for the experiments presented in Chapter 7. Anything appearing after a '#' character is a comment, and spacing and indentation have been added for readability.

```
### ER_200 ###
network_builder main(): (ROOT
   (INIT_NODE (ADD_ALL_NODES))
   (GROW_NODE
      (CONNECT_W_PROB (INDEX_TO_PROB (addIndex (index_1) (index_1))))
      (GROW_NODE (CONNECT_STUB (prob_0.01) (false))
      (EMPTY_GROW_LIST)))
   (FINISH_NODE
      (REMOVE_PROB (INDEX_TO_PROB (index_3)))
      (EMPTY_FINISH_LIST)))
### END ER_200 ###
```

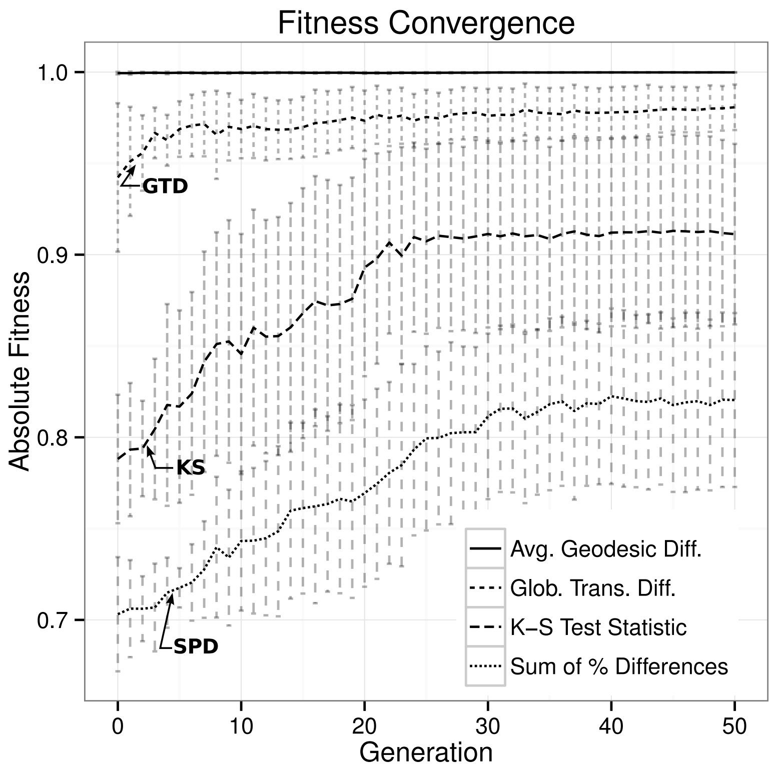


Figure B.10: Watts-Strogatz experiment. Average fitness convergence, over 50 runs.

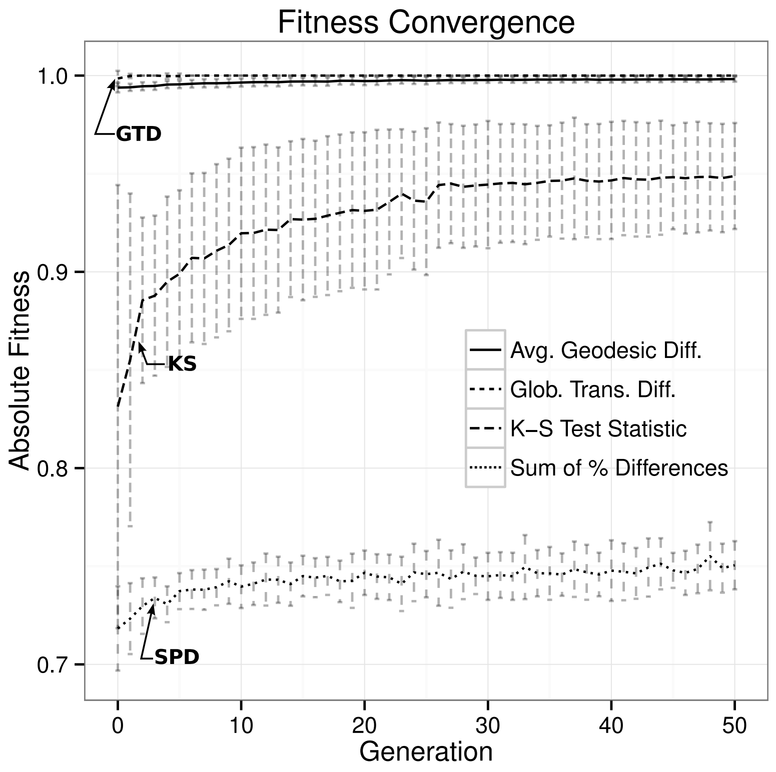


Figure B.11: Barabási-Albert experiment. Average fitness convergence, over 50 runs.

```
### SW_200 ###
network_builder main(): (ROOT
   (INIT_NODE (BUILD_RING))
   (grow_list_t_if (TRUE_WITH_PROB (prob_0.01))
   #then
      (EMPTY_GROW_LIST)
   #else
      (GROW_NODE (CONNECT_STUB (subProb (prob_0.01) (prob_0.01))
         (TRUE_WITH_PROB (prob_0.01)))
      (grow_list_t_if (TRUE_WITH_PROB (INDEX_TO_PROB (index_3)))
      #then
         (GROW_NODE (CREATE_TRIANGLE)
         (EMPTY_GROW_LIST))
   (grow_list_t_if (TRUE_WITH_PROB (prob_0.01))
   #then
      (GROW_NODE (CREATE_TRIANGLE)
      (grow_list_t_if (TRUE_WITH_PROB (prob_0.01))
      #then
         (EMPTY_GROW_LIST)
      #else
         (GROW_NODE (CONNECT_STUB (addProb (prob_0.01)
            (FLOAT_TO_PROB (MAX_DEGREE))) (false))
         (GROW_NODE (CONNECT_RAND)
         (EMPTY_GROW_LIST)))))
      (GROW_NODE (CONNECT_STUB (addProb (prob_0.01)
         (FLOAT_TO_PROB (CURRENT_NODE_DEGREE))) (false))
      (GROW_NODE (CONNECT_RAND) (EMPTY_GROW_LIST))))))
   (EMPTY_FINISH_LIST))
### END SW_200 ###

### BA_200 ###
network_builder main():(ROOT
   (INIT_NODE (SET_GROW_NODES))
   (GROW_NODE
      (CONNECT_STUB_PERSIST
      (INDEX_TO_PROB (index_1)) (true))
      (EMPTY_GROW_LIST))
   (EMPTY_FINISH_LIST))
### END BA_200 ###
```

Appendix C

Application to Cortical Networks: Supplementary Results

Evolution of a Cortical Model Without Community Structure

The GP system was originally applied as it was in Chapter 7 to the cortical data discussed in Chapter 8. The preliminary results showed that the GP system was able to evolve a model able to produce some of the interesting features of the cat cortex, including its unusual degree distribution (see Fig. C.2). The preliminary results also showed that the evolved model was competitive with the Watts-Strogatz (WS) and Barabási-Albert (BA) models in terms of the degree distribution (see Fig. C.1), and some of the properties of interest discussed in Chapter 8. However, plots of the graphs generated by the evolved model (see C.3) indicated the model failed to create the defined communities apparent in the cortical data. These preliminary results motivated a deeper look at creating models with community structure.

The final decision to use a black-box community-detection algorithm to inform the GP system came after a suspicion that the GP language likely cannot handle creating communities like those found in the cortical data on its own (it was never designed to), and given that community detection and evolution was never intended to be within the scope of this thesis the solution proposed in Chapter 8 is somewhat of a compromise. However, the proposed method does provide a useful first step in the direction of evolving models of complex networks with community structure; a significant contribution given that there is no prior work in this area.

GP Trees

This section contains the GP trees evolved by the intercluster and cluster models which were used to construct the final model that generated the results presented in Chapter 8.

```
### Cluster Model ###
network_builder main(): (ROOT
   (INIT_NODE (SET_GROW_NODES))
   (GROW_NODE (CONNECT_W_PROB (FLOAT_TO_PROB 0.985054))
   (GROW_NODE (CONNECT_STUB_PERSIST
      (INDEX_TO_PROB (addIndex (index_1) (index_7))) (true))
   (GROW_NODE (CONNECT_STUB_PERSIST (subProb
      (addProb (INDEX_TO_PROB (index_7)) (subProb
      (subProb (addProb (INDEX_TO_PROB (index_7))
      (prob_0.01)) (addProb (FLOAT_TO_PROB 0.47837)
      (prob_0.01))) (FLOAT_TO_PROB 0.985054)))
      (subProb (prob_0.01) (prob_0.01))) (true))
   (GROW_NODE (CONNECT_STUB_PERSIST (prob_0.01) (true))
   (GROW_NODE (CONNECT_STUB_PERSIST (addProb
      (INDEX_TO_PROB (addIndex (addIndex (index_1)
      (index_3)) (index_3))) (prob_0.01)) (true))
   (GROW_NODE (CONNECT_STUB_PERSIST (addProb
      (FLOAT_TO_PROB (AVG_DEGREE))
      (addProb (FLOAT_TO_PROB (FINAL_VCOUNT))
      (FLOAT_TO_PROB (CURRENT_NODE_DEGREE)))) (true))
   (EMPTY_GROW_LIST)))))))
   (FINISH_NODE (REMOVE_PROB (INDEX_TO_PROB (index_3)))
   (FINISH_NODE (REMOVE_PROB (prob_0.01))))
### END Cluster Model ###

### Intercluster Model ###
network_builder main(): (ROOT
   (INIT_NODE (SET_GROW_NODES))
   (GROW_NODE (CREATE_TRIANGLE)
   (grow_list_t_if (TRUE_WITH_PROB (FLOAT_TO_PROB 0.758118))
   #then
      (grow_list_t_if (< (FINAL_VCOUNT) (CURRENT_NODE_DEGREE))
      #then
         (GROW_NODE (CONNECT_STUB (FLOAT_TO_PROB (AVG_DEGREE)) (false))
         (GROW_NODE (CONNECT_STUB (FLOAT_TO_PROB (TOTAL_VERTEX_COUNT)) (false))
         (GROW_NODE (CREATE_TRIANGLE)
         (GROW_NODE (CONNECT_STUB (FLOAT_TO_PROB (AVG_DEGREE)) (true))
         (GROW_NODE (CREATE_TRIANGLE)
         (GROW_NODE (DUPLICATE (subProb (addProb
            (FLOAT_TO_PROB (AVG_DEGREE)) (prob_0.01))
            (addProb (FLOAT_TO_PROB (TOTAL_VERTEX_COUNT))
            (INDEX_TO_PROB (index_7)))))
         (GROW_NODE (CREATE_TRIANGLE) (EMPTY_GROW_LIST))))))))
      #else
         (GROW_NODE (CONNECT_STUB_PERSIST (subProb
         (addProb (prob_0.01) (prob_0.01))
         (subProb (FLOAT_TO_PROB (FINAL_VCOUNT))
         (addProb (subProb (prob_0.01) (prob_0.01))
         (INDEX_TO_PROB (index_7))))) (false))
         (EMPTY_GROW_LIST)))
   #else
```

```
        (GROW_NODE (CONNECT_STUB_PERSIST (INDEX_TO_PROB
           (subIndex (index_1) (index_7)))
           (TRUE_WITH_PROB (prob_0.01)))
        (GROW_NODE (DUPLICATE (addProb (prob_0.01) (addProb (prob_0.01)
           (subProb (FLOAT_TO_PROB (((TOTAL_EDGE_COUNT) - (FINAL_VCOUNT))
           * (TOTAL_VERTEX_COUNT))) (addProb (addProb (FLOAT_TO_PROB
           (TOTAL_VERTEX_COUNT)) (INDEX_TO_PROB (index_7))) (prob_0.01))))))
        (GROW_NODE (CONNECT_STUB (addProb
           (INDEX_TO_PROB (subIndex (index_1) (index_7)))
           (subProb (INDEX_TO_PROB (index_7))
           (INDEX_TO_PROB (index_1)))) (true))
        (GROW_NODE (CONNECT_RAND)
        (GROW_NODE (CONNECT_STUB_PERSIST (subProb (addProb
           (subProb (subProb (prob_0.01) (prob_0.01))
           (addProb (addProb (addProb (FLOAT_TO_PROB (TOTAL_VERTEX_COUNT))
           (INDEX_TO_PROB (index_7))) (prob_0.01)) (prob_0.01)))
           (FLOAT_TO_PROB (0.758118 - 0.758118)))
           (subProb (addProb (prob_0.01)
           (prob_0.01)) (prob_0.01))) (< (FINAL_VCOUNT) (TOTAL_EDGE_COUNT)))
        (GROW_NODE (CONNECT_W_PROB (prob_0.01))
        (EMPTY_GROW_LIST)))))))))
     (FINISH_NODE (REMOVE_PROB (INDEX_TO_PROB (index_7))))
### END Intercluster Model ###
```

Table C.1: Values of network properties WS, BA, and Evolved models vs. Cortical.

	$\|E\|$	transitivity	diameter	geodesic	communities
WS	520	0.53	2.98	1.61	3.76
BA	506	0.48	2.84	1.62	3.66
Evolved	550	0.52	3	1.59	3
Cortex	515	0.58	3	1.64	3

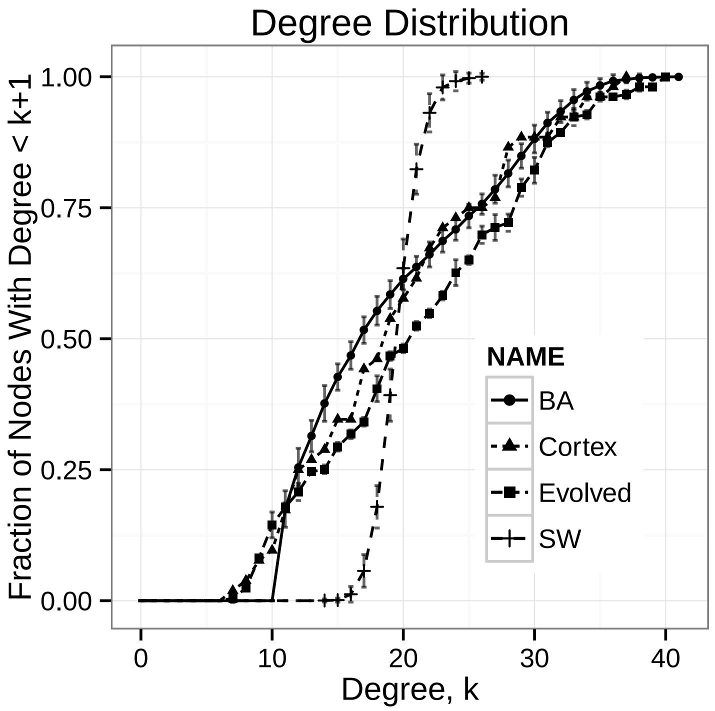


Figure C.1: Comparison of cumulative degree distributions, Evolved, WS, and BA models vs. Cortex.

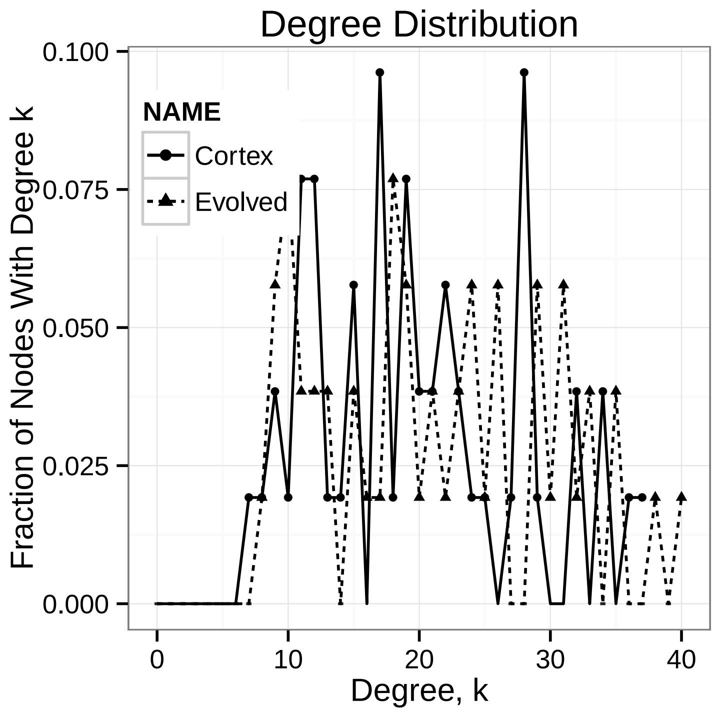


Figure C.2: Comparison of a single degree distribution generated by the evolved model v.s. the Cortex degree distribution.

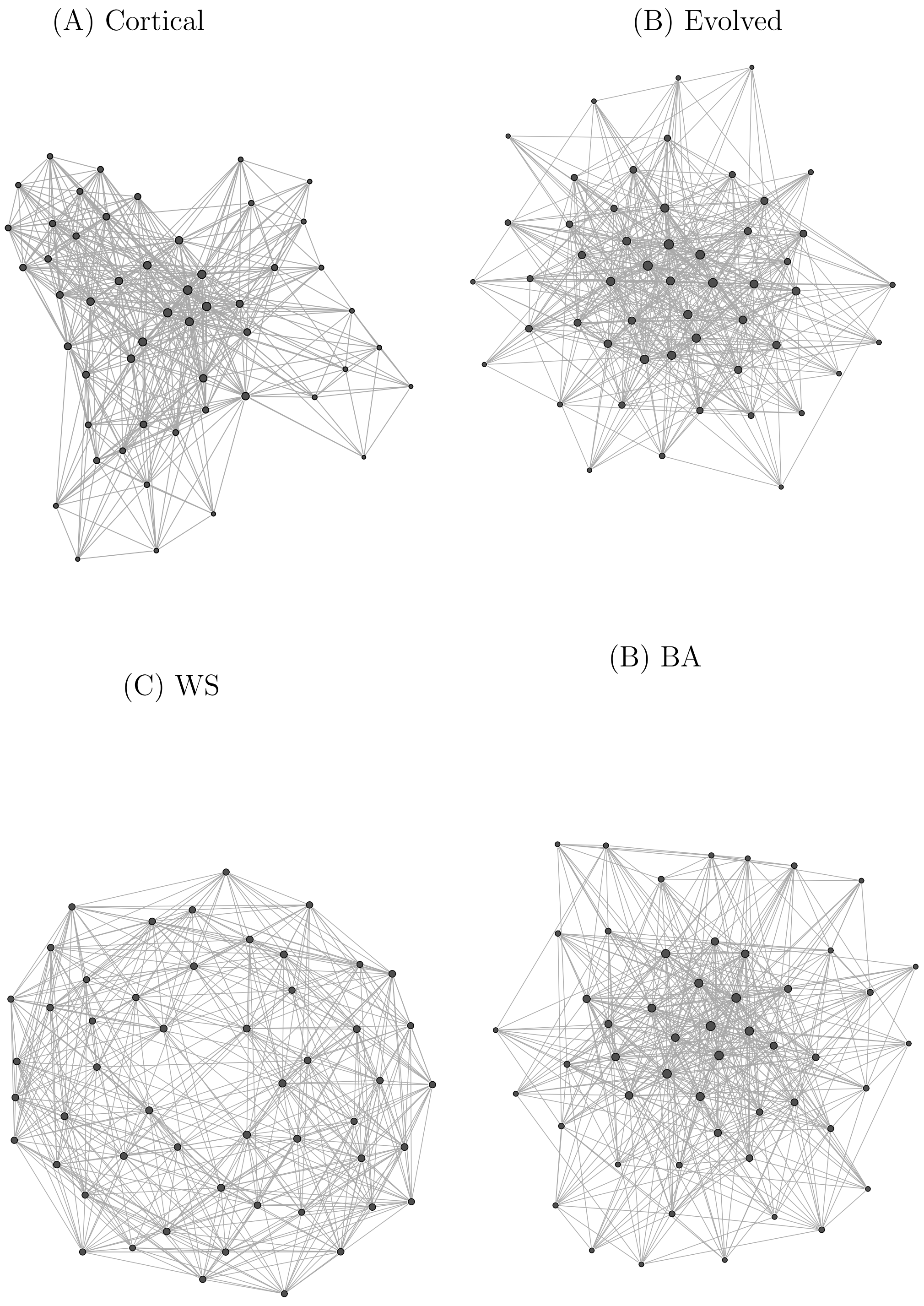


Figure C.3: A graph generated by the evolved model without community structure, as well as the BA and WS models v.s. the cortex data.

Bibliography

[1] L. A. Adamic and B. A. Huberman. Growth dynamics of the world wide web. *Nature*, 401(6749):131, 1999.

[2] L. A. Adamic and B. A. Huberman. Power-law distribution of the world wide web. *Science*, 287(5461):2115, 2000.

[3] R. Albert and A.L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47–97, 2002.

[4] Réka Albert. Scale-free networks in cell biology. *Journal of Cell Science*, 118:4947–4957, 2005.

[5] L. A. N. Amaral, A. Scala, M. Barthélémy, and H. E. Stanley. Classes of small-world networks. *PNAS*, 97(21):11149–11152, 2000.

[6] Gregory M. Ames, Dylan B. George, Christian P. Hampson, Andrew R. Kanarek, Cayla D. McBee, Dale R. Lockwood, Jeffrey D. Achter, and Colleen T. Webb. Using network properties to predict disease dynamics on human contact networks. *Proc. R. Soc. B*, 2011.

[7] A. L. Barabási and R. Albert. Emergence of scaling in random networks. *Science (New York, N.Y.)*, 286(5439):509–512, October 1999.

[8] A. L. Barabasi, H. Jeong, Z. Neda, E. Ravasz, A. Schubert, and T. Vicsek. Evolution of the social network of scientific collaborations. *PHYSICA A*, 311:3, 2002.

[9] Danielle S. Bassett, E. Bullmore, Beth A. Verchinski, Venkata S. Mattay, Daniel R. Weinberger, and Andreas Meyer-Lindenberg. Hierarchical organization of human cortical networks in health and schizophrenia. *The J. of Neurosci.*, 28(37):9239–9248, 2008.

[10] P.J. Bentley and J.P. Wakefield. Finding acceptable solutions in the pareto-optimal range using multiobjective genetic algorithms. In *Soft Computing in Engineering Design and Manufacturing*, pages 231–240. Springer, June 1997.

[11] Johannes Berg, Michael Lassig, and Andreas Wagner. Structure and evolution of protein interaction networks: a statistical model for link dynamics and gene duplications. *BMC Evolutionary Biology*, 4(1):51, 2004.

[12] Phillip Bonacich. Power and centrality: A family of measures. *American Journal of Sociology*, 92(5):1170–1182, 1987.

[13] E. Bullmore and O. Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature Reviews Neuroscience*, 10(3):186–198, 2009.

[14] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, November 2009.

[15] Carlos A. Coello. An updated survey of GA-based multiobjective optimization techniques. *ACM Comput. Surv.*, 32(2):109–143, June 2000.

[16] J Cohen. *Statistical power analysis for the behavioral sciences*, volume 2. Lawrence Erlbaum Associates, 1988.

[17] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006.

[18] A. Davis, B. Gardner, and M. R. Gardner. *Deep South.* University of Chicaco Press, Chicago, 1941.

[19] D. Morton de Lachapelle, D. Gfeller, and P. De Los Rios. Shrinking matrices while preserving their eigenpairs with application to the spectral coarse graining of graphs. *Submitted to SIAM Journal on Matrix Analysis and Applications*, 2008.

[20] F. De Vico Fallani, R. Sinatra, L. Astolfi, D. Mattia, F. Cincotti, V. Latora, S. Salinari, M.G. Marciani, A. Colosimo, and F. Babiloni. Community structure of cortical networks in spinal cord injured patients. In *Proc. IEEE EMBS*, Proc. EMBS, pages 3995 –3998, 2008.

[21] S. N. Dorogovtsev, J. F. F. Mendes, and A. N. Samukhin. Structure of growing networks with preferential linking. *Physical Review Letters*, 85(21):4633–4636, 2000.

[22] Leo Egghe and Ronald Rousseau. *Introduction to Informetrics : Quantitative Methods in Library, Documentation and Information Science.* Elsevier Science Publishers, 1990.

[23] P. Erdös and A. Rényi. On random graphs, i. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[24] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 8:128–140, 1736.

[25] Santo F. Community detection in graphs. *Phys. Reports*, 486(3-5):75 – 174, 2010.

[26] D. A. Fell and A. Wagner. The small world of metabolism. *Nature Biotechnology*, 18(11):1121–1122, 2000.

[27] I. C. R. Ferrer and R. V. Solé. The small world of human language. In *Proceedings. Biological sciences/The Royal Society*, volume 268, pages 2261–2265, 2001.

[28] Robert Flack. The robgp genetic programming system, 2011. http://robgp.sourceforge.net/.

[29] Robert W. J. Flack and Brian J. Ross. Evolution of architectural floor plans. In *Proc. EvoApplications'11*, pages 313–322. Springer-Verlag, 2011.

[30] L. C. Freeman. Some anecedents of social network analysis. *Connections*, 19:39 – 42, 1996.

[31] L.C. Freeman. *The Development Of Social Network Analysis: A Study In The Sociology Of Science*. Empirical Press, BookSurge, 2004.

[32] Linton C. Freeman. Finding social groups: A meta-analysis of the southern women data. In Ronald Breiger, Kathleen Carley, and Philippa Pattison, editors, *Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers*. National Academies Press, 2003.

[33] Thomas M. J. Fruchterman and Edward M. Reingold. Graph drawing by force-directed placement. *Softw. Pract. Exper.*, 21(11):1129–1164, November 1991.

[34] D. Gfeller. *Simplifying Complex Networks: From a Clustering to a Coarse Graining Strategy*. LAP Lambert Acad. Publ., 2009.

[35] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[36] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., 1st edition, 1989.

[37] Anna Goldenberg, Alice X. Zheng, Stephen E. Fienberg, and Edoardo M. Airoldi. A survey of statistical network models. *Foundations and Trends in Machine Learning*, 2(2):129–233, 2009.

[38] Biyu J. He, John M. Zempel, Abraham Z. Snyder, and Marcus E. Raichle. The Temporal Structures and Functional Significance of Scale-free Brain Activity. *Neuron*, 66:353–369, 2000.

[39] Yong He, Alain Dagher, Zhang Chen, Arnaud Charil, Alex Zijdenbos, Keith Worsley, and Alan Evans. Impaired small-world efficiency in structural cortical networks in multiple sclerosis associated with white matter lesion load. *Brain*, 132(12):3366–3379, December 2009.

[40] L. V. Hedges. Distribution theory for Glass's estimator of effect size and related estimators. *Journal of Educational Statistics*, 6(2):107–128, 1981.

[41] S.L. Hill, Wang Y., Riachi I., Schürmann F., and Markram H. Statistical connectivity provides a sufficient foundation for specific functional connectivity in neocortical neural microcircuits. *PNAS*, Sept 2012.

[42] Bernardo A. Huberman. *The Laws of the Web: Patterns in the Ecology of Information*. MIT Press, Cambridge, MA, USA, 2001.

[43] Williams Richard J. and Martinez Neo D. Simple rules yield complex food webs. *Nature*, 404:180–183, 2000.

[44] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A. L. Barabási. The large-scale organization of metabolic networks. *Nature*, 407(6804):651–654, 2000.

[45] Marcus Kaiser, Robert Martin, Peter Andras, and Malcolm P Young. Simulation of robustness against lesions of cortical networks. *Euro. J. of Neurosci.*, 10:3185–3192, Aug 2007.

[46] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, March 1953.

[47] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Sys. Tech. J.*, 49(2):291–308, 1970.

[48] Jon Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. The web as a graph: Measurements, models, and methods. *Computing and Combinatorics*, pages 1–17, 1999.

[49] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, September 1999.

[50] K. Klemm and V.M. Eguiluz. Growing scale-free networks with small-world behavior. *Phys Rev E Stat Nonlin Soft Matter Phys*, 65(5 Pt 2):057102, 2002.

[51] Frank Konietschke. *nparcomp: Perform multiple comparisons and compute simultaneous confidence intervals for the nonparametric relative contrast effects.*, 2012. R package version 2.0.

[52] J. R. Koza. *On the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[53] J. R. Koza. Genetic programming ii: Automatic discovery of reusable subprograms, 1994.

[54] John R. Koza, David Andre, Forrest H. Bennett, and Martin A. Keane. *Genetic Programming III: Darwinian Invention & Problem Solving*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[55] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 177–187, New York, NY, USA, 2005. ACM.

[56] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007.

[57] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[58] Stanley Milgram. The small-world problem. *Psychology Today*, 1(1):61–67, 1967.

[59] R.G. Miller. *Simultaneous Statistical Inference*. Springer series in statistics. Springer-Verlag, 1981.

[60] O. Mokryn, M. Blattner, and Y. Shavitt. The Role of Trends in Evolving Networks. *ArXiv e-prints*, June 2013.

[61] David J. Montana. Strongly typed genetic programming. *Evol. Comput.*, 3(2):199–230, June 1995.

[62] Jose M. Montoya and Ricard V. Solé. Small world patterns in food webs. *Journal of Theoretical Biology*, 214(3):405 – 412, 2002.

[63] J.L. Moreno and H.H. Jennings. *Who shall survive?: A new approach to the problem of human interrelations*. Nervous and mental disease monograph series. Nervous and mental disease publishing co., 1934.

[64] Ted Mouw and Ashton M. Verdery. Network sampling with memory: A proposal for more efficient sampling from social networks. *Sociological Methodology*, 42(1):206–256, 2012.

[65] T. Nepusz, A. Petróczi, L. Négyessy, and F. Bazsó. Fuzzy communities and the concept of bridgeness in complex networks. *Phys Rev E Stat Nonlin Soft Matter Phys*, 77(1 Pt 2):016107, 2008.

[66] Tamás Nepusz, László Négyessy, Gábor Tusnády, and Fülöp Bazsó. Reconstructing cortical networks: Case of directed graphs with high level of reciprocity. In Béla Bollobás, Robert Kozma, and Dezsö Miklós, editors, *Handbook of Large-Scale Random Networks*, volume 18 of *Bolyai Society Mathematical Studies*, pages 325–368. Springer, 2008.

[67] Theoden I. Netoff, Robert Clewley, Scott Arno, Tara Keck, , and John A.White. Epilepsy in small-world networks. *The Journal of Neuroscience*, 24:8075–8083, 2004.

[68] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[69] M E J Newman. Finding community structure in networks using the eigenvectors of matrices. *Phys. Rev. E*, 74(physics/0605087):36104. 21 p, May 2006.

[70] M. E. J. Newman. Internet at the autonomous systems level, July 2006. http://www-personal.umich.edu/ mejn/netdata/.

[71] M. E. J. Newman. Modularity and community structure in networks. *PNAS*, 103(23):8577–8582, 2006.

[72] M. E. J. Newman and M. Girvan. Community structure in social and biological networks. *PNAS*, 99(12):7821–7826, 2002.

[73] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(2):026113, 2004. PT: J; PN: Part 2; PG: 15.

[74] Mark Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010.

[75] L. Négyessy, T. Nepusz, L. Kocsis, and F. Bazsó. Prediction of the main cortical areas and connections involved in the tactile function of the visual cortex by network analysis. *Eur J Neurosci*, 23(7):1919–30, 2006.

[76] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, Stanford University, Stanford, CA, USA, November 1998.

[77] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008. (With contributions by J. R. Koza).

[78] Brenton J. Prettejohn, Matthew J. Berryman, and Mark D. McDonnell. Methods for generating complex networks with selected structural properties for simulations: A review and tutorial for neuroscientists. *Frontiers in Comp. Neurosci.*, 5:11, 2011.

[79] Derek De Solla Price. A general theory of bibliometric and other cumulative advantage processes. *JASIST*, pages 292–306, 1976.

[80] Anatol Rapoport. Contribution to the theory of random and biased nets. *The bulletin of mathematical biophysics*, 19(4):257–277, 1957.

[81] B.J. Ross. Evolution of stochastic bio-networks using summed rank strategies. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 773–780, 2011.

[82] S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2010.

[83] Shuzo Sakata, Yusuke Komatsu, and Tetsuo Yamamori. Local design principles of mammalian cortical networks. *Neuroscience Research*, 51(3):309 – 315, 2005.

[84] Marcel Salathé, Maria Kazandjieva, Jung Woo Lee, Philip Levis, Marcus W. Feldman, and James H. Jones. A high-resolution human contact network for infectious disease transmission. *PNAS*, 2010.

[85] J. W. Scannell, C. Blakemore, and M. P. Young. Analysis of connectivity in the cat cerebral cortex. *J. of Neurosci.*, 15:1463, 1995.

[86] John P. Scott. *Social Network Analysis: A Handbook*. SAGE, Thousand Oaks, CA, 2000.

[87] Ray Solomonoff and Anatol Rapoport. Connectivity of random nets. *B. Math. Biophys.*, 13:107–117, 1951.

[88] Olaf Sporns, Christopher J. Honey, and Rolf Kötter. Identification and classification of hubs in brain networks. *PLoS ONE*, 2(10):e1049, 10 2007.

[89] Olaf Sporns and JonathanD. Zwi. The small world of the cerebral cortex. *Neuroinformatics*, 2:145–162, 2004.

[90] Cornelis Stam and Jaap Reijneveld. Graph theoretical analysis of complex networks in the brain. *Nonlinear Biomedical Physics*, 1(1):3, 2007.

[91] Cornelis Jan Stam and Eveline Astrid de Bruin. Scale-free dynamics of global functional connectivity in the human brain. *Human Brain Mapping*, 22(2):97–109, 2004.

[92] OpenFlights team. Openflights flight database, Oct. 2012. http://openflights.org/.

[93] A Vazquez, A Flammini, A Maritan, and A Vespignani. Modeling of protein interaction networks. *ComplexUs*, 1:38–44, 2003.

[94] M. Ventresca and D. Aleman. Evaluation of strategies to mitigate contagion spread using social network characteristics. *Social Networks*, 2013. (in press).

[95] S. Wasserman and K. Faust. *Social network analysis*. Cambridge University Press, Cambridge, 1994.

[96] Duncan J. Watts. Networks, dynamics and the small world phenomenon. *AJS*, 105:493–527.

[97] Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[98] S. H. Yook, H. Jeong, A. L. Barabási, and Y. Tu. Weighted evolving networks. *Physical Review Letters*, 86(25):5835–5838, 2001.

[99] S. H. Yook, H. Jeong, and A. L. Barabási. Modeling the internet's large-scale topology. *Proceedings of the National Academy of Sciences*, 99(21):13382, 2002.

[100] W.W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.

[101] G. Zamora-López, C. Zhou, and J. Kurths. Graph analysis of cortical networks reveals complex anatomical communication substrate. *Chaos*, 19(1):015117, 2009.

[102] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: empirical results. *Evolutionary Computation*, 8(2):173–195, 2000.